Tomer Alon

CS361

Programming Languages Principles and Implementation

SML and PROLOG

**SML**

1. What are the types of the following expressions?

* [(1,5), (2,3), (5,6)];
  + **val it = [(1,5), (2,3), (5,6)]: (int \* int) list**
* fun f(x:real) = true;
  + **val f = fn: real -> bool**
* map f;
  + **val it = fn : real list -> bool list**

1. Provide expressions of the following types:

* int \* bool
  + **(1,true)**
* int list \* bool
  + **([1,2,3],true)**
* int \* real -> bool list
  + **fun f(x:int, y:real) = [true, false];**

1. Write the following SML functions:

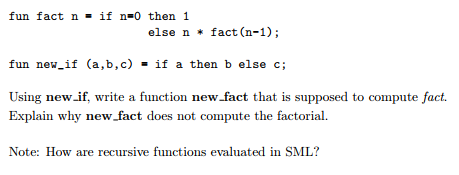


**fun f(n) = if n = 0 then 1 else 2\*f(n-1);**

**or**

**fun f(0) = 1 | f(n) = 2 \* f(n-1);**

**val f = fn : int -> int**



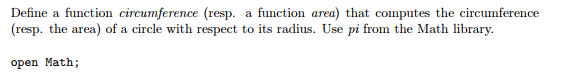
**fun new\_if(a,b,c) = if a then b else c;**

**val new\_if = fn : bool \* 'a \* 'a -> 'a**

**fun new\_fact(n) = new\_if(n = 0, 1, n\*new\_fact(n-1));**

**val new\_fact = fn : int -> int**

**SML evaluates new\_fact(n) with an innermost evaluation, thus not allowing the evaluation to ever end, as opposed to an outermost evaluation, which upon reaching the base case (when n=0) would terminate the evaluation and give a result.**



**open Math;**

**fun circumference(r:real) = 2.0\*pi\*r;**

**val circumference = fn : real -> real**

**fun area(r:real) = pi\*(pow(r,2.0));**

**val area = fn : real -> real**

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

**map (fn x => x + 3) [1,2,3];**

**- val it = [4,5,6] : int list**



**fun move(l) = tl l @ [hd l];**

**move([1,2,3,4,5]);**

**- val move = fn : 'a list -> 'a list**

**val it = [2,3,4,5,1] : int list**

**or**

**fun move([]) = [] | move(l) = tl l @ [hd l];**

**move([1,2,3,4,5]);**

**- val move = fn : 'a list -> 'a list**

**val it = [2,3,4,5,1] : int list**

1. 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.

**Code:**

**datatype 'a BinaryTree = btempty | bt of 'a \* 'a BinaryTree \* 'a BinaryTree;**

**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));**

**fun inorder (btempty) = [] | inorder(bt(root:'a, left, right)) =**

**inorder(left) @ (root :: inorder(right));**

**fun preorder (btempty) = [] | preorder(bt(root:'a, left, right)) =**

**root :: (preorder(left) @ preorder(right));**

**fun postorder (btempty) = [] | postorder(bt(root:'a, left, right)) =**

**(postorder(left) @ postorder(right)) @ (root :: []);**

**fun left\_subtree btempty = btempty | left\_subtree(bt(\_,left,\_)) = left;**

**fun right\_subtree btempty = btempty | right\_subtree(bt(\_,\_,right)) = right;**

**exception label\_has\_nil\_argument;**

**fun label btempty = raise label\_has\_nil\_argument | label(bt(value,\_,\_)) = value;**

**val Tree = bt(2,btempty, bt(3,btempty, bt(7,bt(6,bt(5,btempty,btempty),btempty),**

**bt(8,btempty,btempty))));**

**(\* More sample binary trees \*)**

**val Tree1 = bt(3,btempty,btempty);**

**val Tree2 = bt(5,bt(1,btempty,btempty),btempty);**

**val Tree3 = bt(7,bt(4,btempty,btempty), bt(12,btempty,btempty));**

**val Tree4 = bt("\*", bt("/", bt("-",bt("7",btempty,btempty),**

**bt("a",btempty,btempty)), bt("5",btempty,btempty)), bt("exp",**

**bt("+",bt("a",btempty,btempty), bt("b",btempty,btempty)),**

**bt("3",btempty,btempty)));**

**lookup(Tree,6);**

**lookup(Tree,1);**

**lookup(Tree,9);**

**lookup(Tree3,12);**

**lookup(btempty,6);**

**inorder(Tree);**

**preorder(Tree);**

**postorder(Tree);**

**inorder(Tree4);**

**label(Tree);**

**right\_subtree(Tree);**

**left\_subtree(Tree);**

**(\* Traversal of expressions \*)**

**val Expression = bt("+", bt("\*", bt("+", bt("2",btempty,btempty),**

**bt("5",btempty,btempty)), bt("\*", bt("3",btempty,btempty),**

**bt("4",btempty,btempty))), bt("\*", bt("1",btempty,btempty),**

**bt("6",btempty,btempty)));**

**val Expression2 = bt("+", bt("\*", bt("+", bt("2",btempty,btempty),**

**bt("5",btempty,btempty)), bt("\*", bt("3",btempty,btempty),**

**bt("4",btempty,btempty))), bt("\*", bt("1",btempty,btempty),**

**bt("6",btempty,btempty)));**

**inorder(Expression);**

**preorder(Expression);**

**postorder(Expression);**

**inorder(Expression2);**

**preorder(Expression2);**

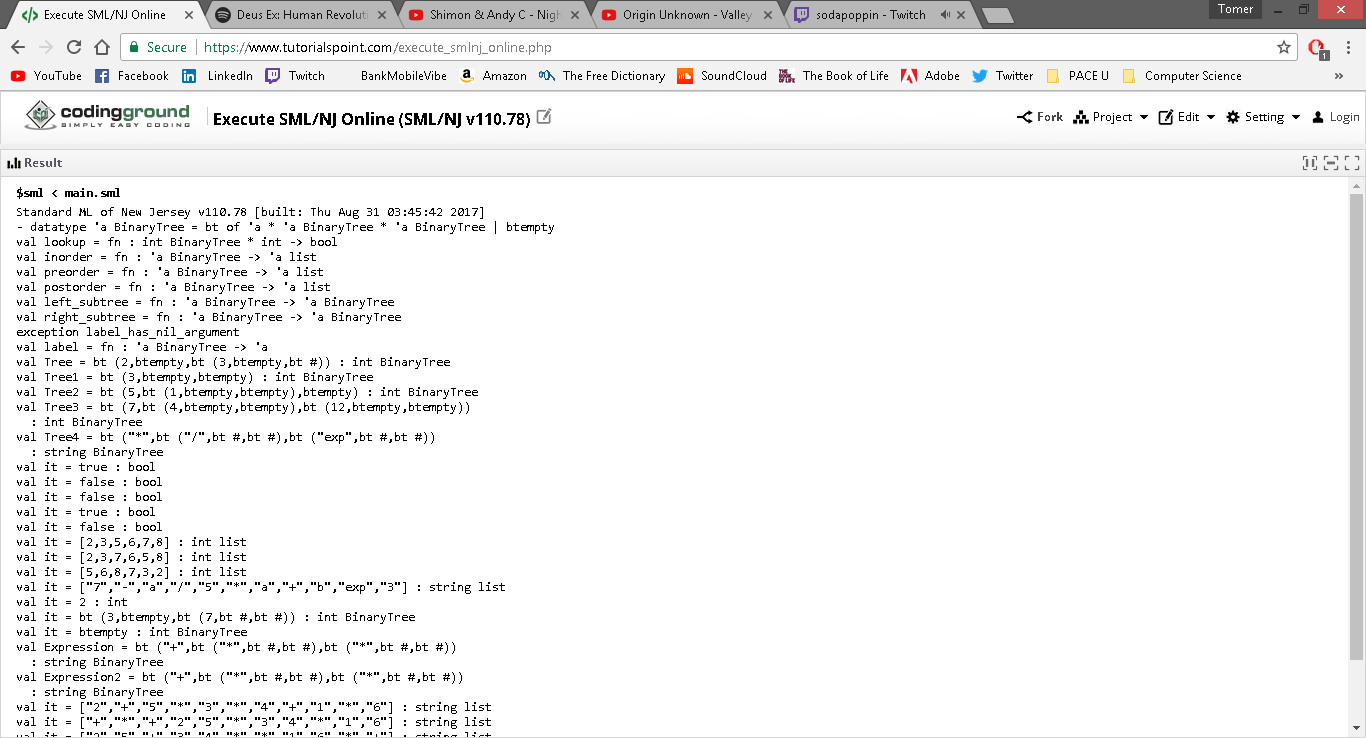
**postorder(Expression2);**

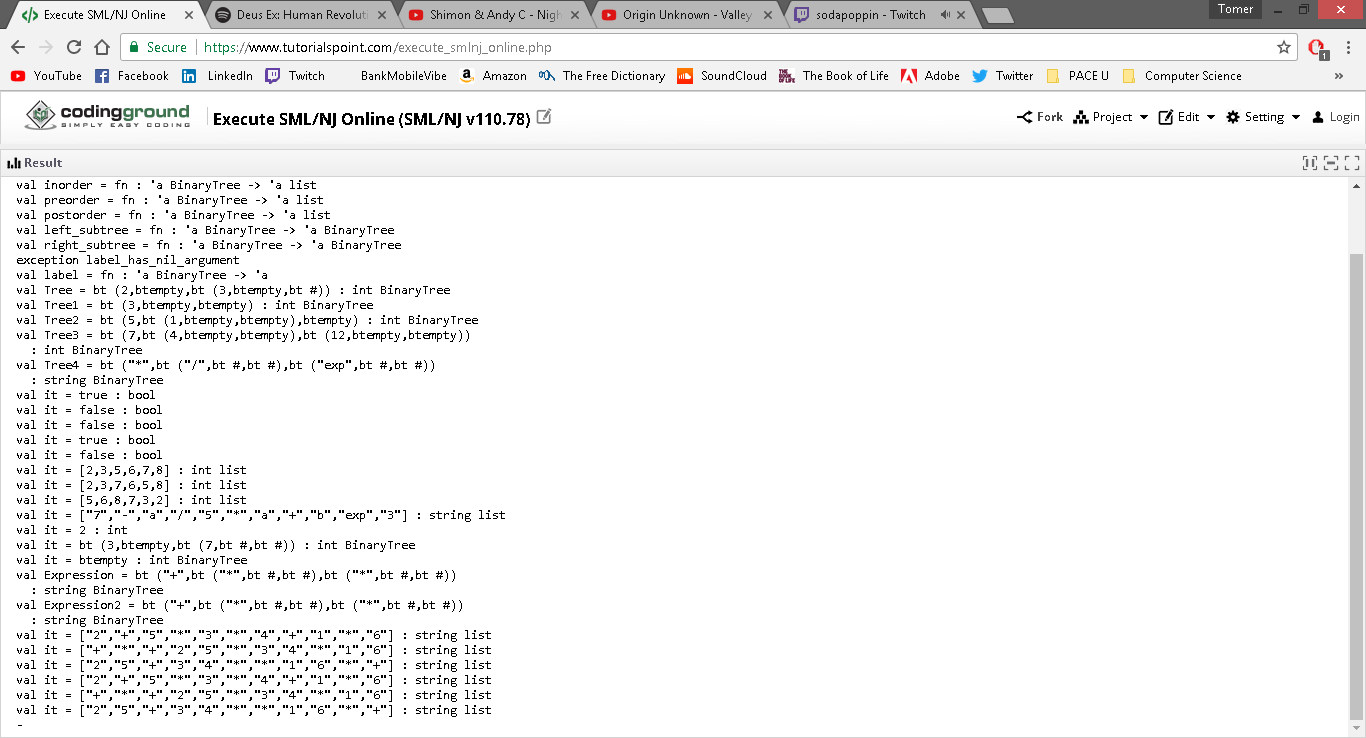






**Output:**





**PROLOG**

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

**mother(linda, paul).**

**mother(cathy, andrew).**

**mother(cathy, laura).**

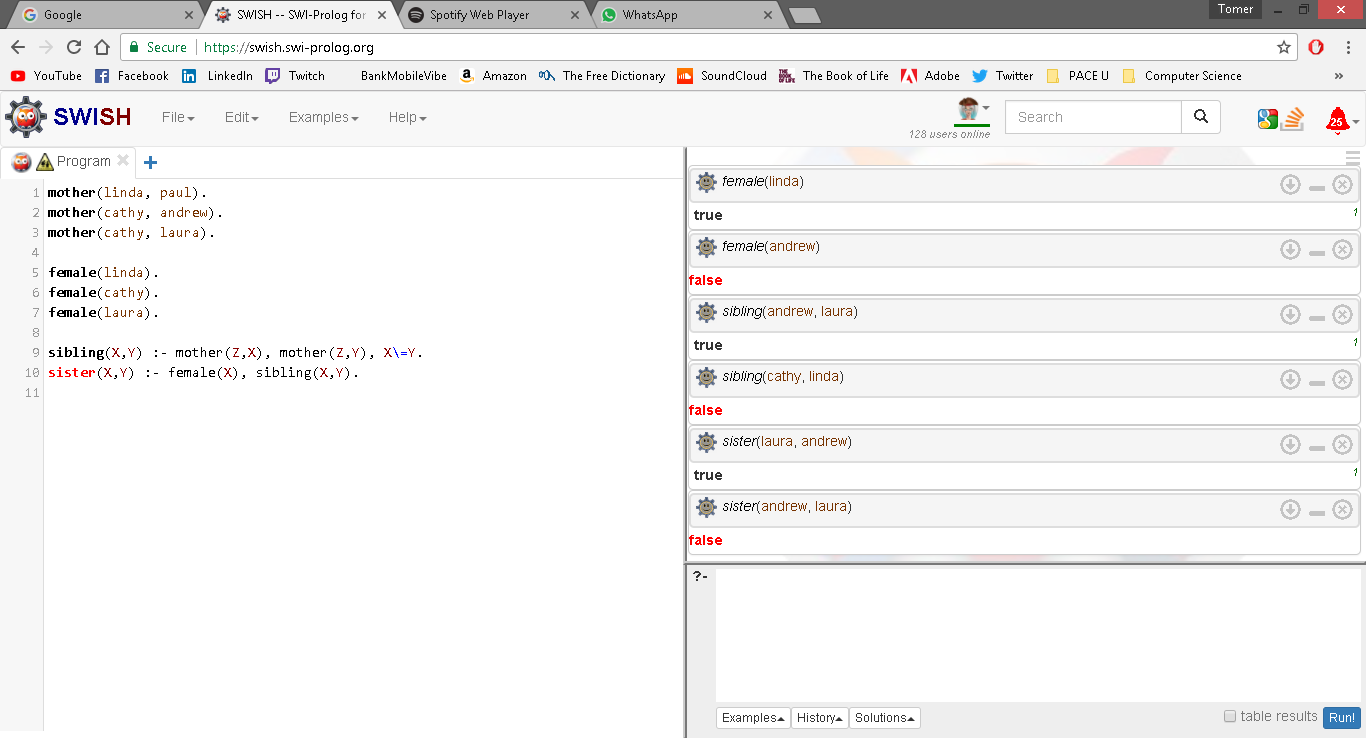
* Define a predicate female(X) which holds iff X is a female
  + **female(linda).**
  + **female(cathy).**
  + **female(laura).**
* Define a predicate sister(X,Y) which holds iff X and Y are sisters
  + Based on the given information, which solely says that Linda and Cathy are mothers, it cannot be determined who are sisters. What *could* be inferred from the given information, however, is whether two people are siblings. In this case, Andrew and Laura can be said to be siblings.

**sibling(X,Y) :- mother(Z,X), mother(Z,Y), X\=Y.**

Nevertheless, we could say that Laura is technically Andrew’s sister.

**sister(X,Y) :- female(X), sibling(X,Y).**

* Implement female and sister in PROLOG
* Provide screenshots



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

**g(X,Y) :- Y is X + 5.**

**?- g(5,Y)**

**Y = 10**