In the name of God

PL homework #6

Seyed Mohammad Mehdi Ahmadpanah – 9031806

from : Concepts in Programming Languages – Mitchell

5.1)
Procedure Q can be declared as:

```
proc Q(x)
real x; begin
real res := x + 1;
Q := res end;
```

In this way, P need a 'procedure' for its argument, and it is possible that the given argument is a procedure, so the procedure call is correct, but the type of second procedure wasn't declared.

5.2)
In this program

```
begin
        integer i;
        integer array A[1:2];
        procedure P(x);
                integer x;
                begin
                        i:=x;
                        x:=i
                end
        i:=1;
        A[1] := 2; A[2] := 3;
        P (A[i]);
        print (i, A[1], A[2])
end
```

in this program when we call p(A[i]) program insert instead of x, A[i]
so we have:
i:=A[i] and A[i] :=i
when we call print (i,A[1],A[2]) program goes and compute this two assignment.
So we have
From this i :=A[i] then i = A[1] → i=2
And from this A[i] :=i then A[2]=2
So print (i,A[1],A[2]) show in the output 2 2 2.

5.3)

a) we can use if clause in this situation:
- fun f(x,y) = if(y=0) then x else if(x=0) then y else x+y;
val f = fn: int * int -> int
b) No. The interpreter returns an error saying that x is already bounded:
stdIn:10.5-10.37 Error: duplicate variable in pattern(s): x (smlnj-ver110.73)
c) First we divide ML function containing patterns into states. Our function using if then else and destructors is based on below form:
if(state 1 happens) then e1
else if(state 2 happens) then e2
else
In every if expression, we will write an expression that checks any attribute or value of variable which will show the state(i) is happening right now. If the result of evaluation is true we will evaluate e1 and return the result. Otherwise, will go to the else part and test the conditions of state(i+1).
The result of the function in part b will be:
- fun eq(x,y) = if(#1(x,y) = #2(x,y)) then true else false;
val eq = fn : ''a * ''a -> bool
d) Because ML does not support equality checking of functions. ML can't do that because:
1. They don't always have the same type. For example, function fun f(x,y) =x; val f = fn: 'a * 'b -> 'a has 'a type which can be any data type.
2. Even if they have the same type, these functions might behave differently and it doesn't seem practical to check their behavior statically.
So in patterns we can't have the same variables because we aren't able to check equality of functions statically.

5.4)

a) Function declaration:
- datatype 'a tree = leaf of 'a | node of 'a tree * 'a tree;
datatype 'a tree = leaf of 'a | node of 'a tree * 'a tree
- fun maptree (x,leaf(y)) = leaf(x(y)) | maptree (x,node(y,z)) =
node(maptree(x,y),maptree(x,z));
val maptree = fn : ('a -> 'b) * 'a tree -> 'b tree
Some Sample:
- val t= node(leaf(4),leaf(3));
val t = node (leaf 4,leaf 3) : int tree
- fun f(x)=x+4;
val f = fn : int -> int

- maptree(f,t);
val it = node (leaf 8,leaf 7) : int tree
- maptree ((fn x=>x+4),t);
val it = node (leaf 8,leaf 7) : int tree
b) val maptree = fn:('a -> 'b) * 'a tree -> 'b tree
ML declare principal types for expressions and in this situation we haven't any limitation for typing of this expression, so ML give a principal type for it.

5.5)

5.6)

a) Curry:
- val curry = fn f => fn x => fn y => f(x,y);
val curry = fn : ('a * 'b -> 'c) -> 'a -> 'b -> 'c
Uncurry:
- val uncurry = fn f => fn (x, y) => f x y;
val uncurry = fn : ('a -> 'b -> 'c) -> 'a * 'b -> 'c
b) according to Curry and Uncurry those are true and their argument types are correct.
uncurry(curry('a * 'b -> 'c)) = uncurry('a -> 'b -> 'c) = 'a * 'b -> c = f curry(uncurry('a -> 'b -> 'c)) = curry('a * 'b -> 'c) = ('a -> 'b -> 'c) = g

5.7)

a) The expression and system may not work because, the "if clause" can involve the data which are not declared in context of the program; so, the compiler won't recognize what value of the union type is valid (integer or string).
In C the compiler , first Checks the program syntactically, then checks it semantically. Since this semantics checking system is applied to all of the program in once; therefore, its semantics won't be able to recognize the problem statically.
The runtime system can't catch these problems because the integer type of the union type exists but its value isn't valid, if the if clause value is evaluated as false.

b) that won't happen in ML:
datatype intString = tagint of int | tagstring of string;
val d=2; val c=1; val x = if c=d
then tagint(3) else tagstring("hello");
let val tagint(m) = x in m+5 end;

datatype intString = tagint of int | tagstring of string
val d = 2 : int

val c = 1 : int
val x = tagstring ”hello” : intString
uncaught exception Bind [nonexhaustive binding failure]
raised at: stdIn:69.135-69.148
This is due to ML’s read-eval-printnature.

5.8)