PL Assignment

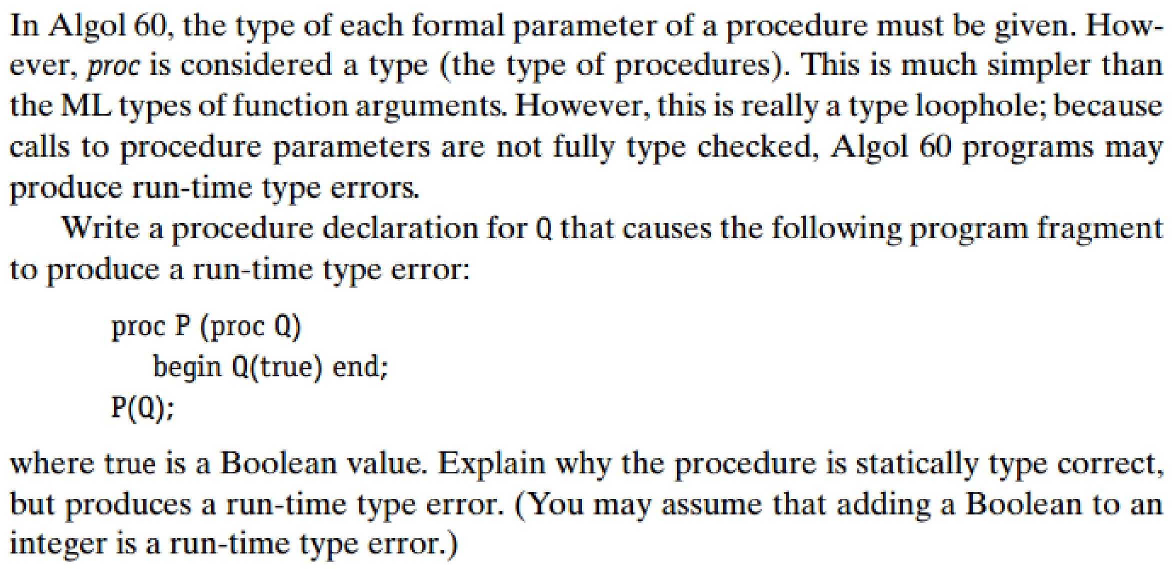
Arash Yadegari (94131092)

December 22, 2016

Concepts in Programming Languages

Chapter 5, Page 122

Problem 5.1 Algol 60 Procedure Types. :



In Algol 60, the type of each formal parameter of a procedure must be given. However, *proc* is considered a type (the type of procedures). This is much simpler than the ML types of function arguments. However, this is really a type loophole; because calls to procedure parameters are not fully type checked, Algol 60 programs may produce run-time type errors.

Write a procedure declaration for Q that causes the following program fragment to produce a run-time type error:

```
proc P (proc Q)
    begin Q(true) end;
P(Q);
```

where true is a Boolean value. Explain why the procedure is statically type correct, but produces a run-time type error. (You may assume that adding a Boolean to an integer is a run-time type error.)

Solution :

```
proc Q(integer b)
        begin print(b+1) end;
```

Procedure 'P' is statically type correct because it gets a procedure and 'Q' is a procedure. Also Q is type correct as it gets an integer parameter and prints it. But it produces a run-time type error as 'Q' procedure is called with a boolean type and it is declared to get an integer parameter.

Problem 5.3 Nonlinear Pattern Matching. :
ML patterns cannot contain repeated variables. This exercise explores this language design decision.

A declaration with a single pattern is equivalent to a sequence of declarations using destructors. For example,

```
val p = (5,2);
val (x,y) = p;
```

is equivalent to

```
val p = (5,2);
val x = #1(p);
val y = #2(p);
```

where #1(p) is the ML expression for the first component of pair p and #2 similarly returns the second component of a pair. The operations #1 and #2 are called *destructors* for pairs.

A function declaration with more than one pattern is equivalent to a function declaration that uses standard if-then-else and destructors. For example,

```
fun f nil = 0
|   f (x::y) = x;
```

is equivalent to

```
fun f(z) = if z=nil then 0 else hd(z);
```

where hd is the ML function that returns the first element of a list.
Questions:

(a) Write a function declaration that does not use pattern matching and that is equivalent to

```
fun f (x,0) = x
|   f (0,y) = y
|   f (x,y) = x+y;
```

ML pattern matching is applied in order, so that when this function is applied to an argument (a, b), the first clause is used if b = 0, the second clause if b ≠ 0 and a=0, and the third clause if b ≠ 0 and a ≠ 0.

(b) Does the method you used in part (a), combining destructors and if-then-else, work for this function?

```
fun eq(x,x) = true
|   eq(x,y) = false;
```

(c) How would you translate ML functions that contain patterns with repeated variables into functions without patterns? Give a brief explanation of a general method and show the result for the function eq in part (b).

(d) Why do you think the designers of ML prohibited repeated variables in patterns? (*Hint:* If f, g : int → int, then the expression f = g is not type-correct ML as the test for equality is not defined on function types.)

Solution :

(a) fun f(a,b) = if b=0 then a else if a=0 then b else plus(a,b)

(b) No, it doesn't work as the signature of function must be just one.

(c) We declare function with different parameters but use if clause for equal parameters. for example for eq(x,x) = true we use fun eq(x,y) = if x=y then true

(d) As mentioned in problem statement, test for equality is not defined on function types in ML as we have type variables that can accept any type and functions' behavior cannot be statically checked. So repeated variables are prohibited in patterns.

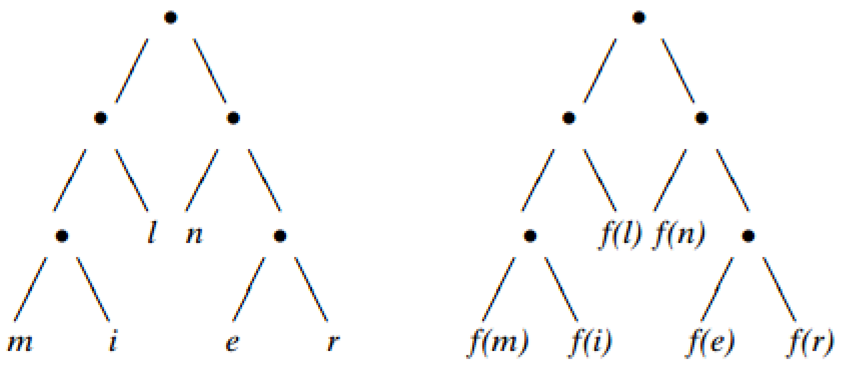
Problem 5.4 ML Map for Trees. :

(a) The binary tree data type

```
datatype 'a tree = LEAF of 'a |
                   NODE of 'a tree * 'a tree;
```

describes a binary tree for any type, but does not include the empty tree (i.e., each tree of this type must have at least a root node).

Write a function maptree that takes a function as an argument and returns a function that maps trees to trees by mapping the values at the leaves to new values, using the function passed in as a parameter. In more detail, if f is a function that can be applied to the leaves of tree t and t is the tree on the left, then maptree f t should result in the tree on the right:



For example, if f is the function fun f(x)=x+1 then

```
maptree f (NODE(NODE(LEAF 1,LEAF 2),LEAF 3))
```

should evaluate to `NODE(NODE(LEAF 2,LEAF 3),LEAF 4)`. Explain your definition in one or two sentences.

(b) What is the type ML gives to your function? Why is it not the expected type ('a → 'a) → 'a tree → 'a tree?

Solution :

(a) datatype 'a tree = leaf of 'a | node of 'a tree * 'a tree;
fun maptree (f,leaf(x)) = leaf(f(x)) | maptree (f,node(x,y)) = node(maptree(f,x),maptree(f,y));

(b) result is : val maptree = fn : ('a -> 'b) * 'a tree -> 'b tree
because f is a general function and output of that is not the same as input('a). It considers the most general type.

Problem 5.6 Currying. :

This problem asks you to show that the ML types $'a \to ('b \longrightarrow 'c)$ and $('a * 'b) \longrightarrow 'c$ are essentially equivalent.

(a) Define higher-order ML functions

Curry: $(('a * 'b) \longrightarrow 'c) \longrightarrow ('a \longrightarrow ('b \longrightarrow 'c))$

and

UnCurry: $('a \longrightarrow ('b \longrightarrow 'c)) \longrightarrow (('a * 'b) \longrightarrow 'c)$

(b) For all functions $f : ('a * 'b) \longrightarrow 'c$ and $g : 'a \longrightarrow ('b \longrightarrow 'c)$, the following two equalities should hold (if you wrote the right functions):

UnCurry(Curry(f)) = f
Curry(UnCurry(g)) = g

Explain why each is true for the functions you have written. Your answer can be three or four sentences long. Try to give the main idea in a clear, succinct way. (We are more interested in insight than in number of words.) Be sure to consider termination behavior as well.

Solution :

(a)
```
- val Curry = fn f => fn a => fn b => f(a,b);
val Curry = fn : ('a * 'b -> 'c) -> 'a -> 'b -> 'c
- val Uncurry = fn f => fn (a, b) => f a b;
val Uncurry = fn : ('a -> 'b -> 'c) -> 'a * 'b -> 'c
```

(b) Uncurry(Curry(f)) = Uncurry(Curry(('a*'b)->'c)) = Uncurry('a->('b->'c)) = ('a*'b)->'c = f
Curry(Uncurry(g)) = Curry(Uncurry('a->('b->'c))) = Curry(('a*'b)->'c) = 'a->('b->'c) = g

Problem 5.7 Disjoint Unions. :

A *union type* is a type that allows the values from two different types to be combined in a single type. For example, an expression of type union(A, B) might have a value of type A or a value of type B. The languages C and ML both have forms of union types.

(a) Here is a C program fragment written with a union type:

```
...
union IntString {
   int i;
   char *s;
} x;
int y;
if ( ... ) x.i = 3 else x.s = "here, fido";
...
y = (x.i) + 5;
...
```

A C compiler will consider this program to be well typed. Despite the fact that the program type checks, the addition may not work as intended. Why not? Will the run-time system catch the problem?

(b) In ML, a union type union(A,B) would be written in the form datatype UnionAB = tag_a of A | tag_b of B and the preceding if statement could be written as

```
datatype IntString = tag_int of int | tag_str of string;
...
val x = if ... then tag_int(3) else tag_str("here, fido");
...
let val tag_int (m) = x in m + 5 end;
```

Can the same bug occur in this program? Will the run-time system catch the problem? The use of tags enables the compiler to give a useful warning message to the programmer, thereby helping the programmer to avoid the bug, even before running the program. What message is given and how does it help?

Solution :

(a) The addition may not work as intended. Because of there is 'if' clause, the value of 'x' variable is unknown(it can be string or integer). Run-time system will not catch the problem because type of 'x' is union of integer and string and it's OK but the value is not valid.

(b)
```
- val x = if false then tag_int(3) else tag_str("here, fido");
val x = tag_str "here, fido" : IntString
- let val tag_int (m) = x in m + 5 end;
uncaught exception Bind [nonexhaustive binding failure]
raised at: stdIn:17.9-17.24
```
As ML has read-eval-print environment the same bug will not occur and run-time system catches the problem.