Principles of Programming Languages - Quiz II (Solutions)

K. Subramani
LDCSEE,
West Virginia University,
Morgantown, WV
{ksmani@csee.wvu.edu}

1 Problems

1. **Type-Checking:**

   Consider the following ML function definition:

   ```
   > fun thrice f  x = f(f(f(x)));
   ```

   Use the Hindley-Milner type-checking algorithm (or any logical procedure) to deduce the type of *thrice()*. You are required to determine the most general type.

   Solution: Let a denote the type of x and $a \rightarrow b$ denote the type of $f()$. Since $f()$ is being applied to a value returned by $f()$, it follows that the return type of $f()$ is identical to the type of its input. Thus $f()$ has type $a \rightarrow a$. The function *thrice()* takes two parameters, viz., a function of type $a \rightarrow a$ and a variable of type a. Applying $f()$ to x (twice) results in value of type a.

   Accordingly, the type of *thrice()* is $(a \rightarrow a) \rightarrow a \rightarrow a$ in curried form, or $(a \rightarrow a) * a \rightarrow a$ in uncurried form. □

2. **Expressions and Statements:**

   (i) Explain the difference(s) between the **if**-expression and **if**-statement in **C**.

   (ii) Given the semantics of the assignment statement in **C**, will the following fragment of code work? Can it be made to work? Justify your answer.

   ```
   (a > b)? (a=3): (b=4);
   ```

   Solution:

   (i) The **if**-expression in **C** is similar to any other function call in that the **if** is actually an operator with three operands, viz., the conditional expression, the "then" expression and the "else" expression. The principal distinction between the **if** operator and other operators in **C**, is that the **if** operator uses delayed evaluation. It is important to note that the **if** expression is concerned only with *returning* a value and that no side-effects are involved.

   On the other hand, the semantics of **if** statement permits multiple side-effects on both the **then** side and the **else** side.

   (ii) Although as per the assignment semantics in **C**, assignment is an operator and returns the assigned value, the given fragment will not work. It cannot be made to work because of type incompatibilities; for instance, the **then** part could involve a float assignment, thereby returning a float, while the **else** part could involve an integer assignment, thereby returning an integer.

   □

3. **Procedures and Environments:**

Consider the following **C** program:

```
int i;
int b[5];

void q (int x)
{
  i++;
  x++;
}

main()
{
  i=1;
  b[1]=3;
  b[2]=4;
  q(b[i]);
  printf(``%d \n'',b[i]);
}
```

What value will be printed assuming that **C** uses the following parameter passing mechanisms: (i) Pass by value, (ii) Pass by value-result, (iii) Pass by name.

Solution: Regardless of whether $b[1]$ is passed by value or value-result, the value that is printed out is $b[2]$, since the global variable i is modified within $q()$. Thus, under the first two mechanisms, the value 4 is printed out, although $b[1]$ is 3 under pass by value and 4 under pass by value-result.

The pass by name mechanism involves textual replacement, i.e., $x++$ is replaced by $b[i]++$. Thus, $b[2]$ is now changed to 5, which is the value printed out. Note that $b[1]$ is unaltered. □

4. **Scheme programming:**

Write a function in SCHEME that takes as input two *sorted* integer lists L and M and returns a list obtained by *merging* L and M. You may assume that the lists are sorted in ascending order.

Solution: The following function is one approach:

```
(define (merge L M)
  (cond ((null? M) L)
        ((null? L) M)
        ((<= (car L) (car M)) (cons (car L) (merge (cdr L) M)))
        (else (cons (car M) (merge L (cdr M))))))
```

□

5. **ML programming:**

(i) Describe how you would declare a type for Binary Search Trees on integers in ML.

(ii) Write a function named PRE-TRAVERSE(), which takes as input a Binary Search Tree of the form described above and outputs the list of elements obtained by a *pre-order* traversal of this tree.

Solution:

(i) `datatype` *int* `BST= NIL | Node of` *int* $*$ *int* `BST`$*$*int* `BST;`

(ii)
```
fun Pre-Traverse Nil = []
  | Pre-Traverse(Node(d,l,r)) = [d]@(Pre-Traverse l)@(Pre-Traverse r);
```

□