Midterm : CS 2104 : Programming Language Concepts

September 20, 10 - 11 AM

Instructions to Candidates

- Answer **ALL** questions. This exam has two questions.
- All answers **MUST** come with the correct explanations. There is no credit for guessing. A correct answer without the correct explanation will receive no marks.
- Answers must be written in the space provided in this booklet;
  otherwise they will not be graded.
- This is an **OPEN BOOK** examination. You are allowed to bring in any books/lecture notes etc., but not a laptop.
- You can ask for extra sheets for rough work.
- **PLEASE WRITE YOUR MATRICULATION NUMBER BELOW**.

MATRICULATION NO.:

(This portion is reserved for the examiner's use only)

Question		Marks
Question A	14	
Question B	11	
TOTAL	25	

Section A: Language Syntax and Semantics
(2 + 3 + 4 + 2 + 3) = 14 marks

1. The programming language IMP was discussed in class. Consider the IMP program :

   ```
   while (a = a) do a := 1
   ```

   Assume a is a variable which is initialized to 0. Then, starting from this initial state how many states in total are reached by execution of this program ? Explain your answer.

   ANSWER: The only variable is a; the valuations of a define the states. Two states are traversed {a = 0, a= 1}

2. Show that the following BNF grammar is ambiguous.

$$S \rightarrow SS \mid a \mid b$$

Then construct an *equivalent* BNF grammar (which has the same language) which is unambiguous. [1 + 2] = 3 marks

answer:

Any string with > 2 characters accepted by the grammar will have more than one parse tree *e.g.* the string abb This is because the rule S -¿ SS partitions the string into two parts and there are several ways of partitioning it.

An equivalent unambiguous grammar is

$$S \rightarrow aS \mid bS \mid a \mid b$$

3. Consider the following BNF grammar of algebraic expressions with a single identifier a

$$S \rightarrow S + S \mid S - S \mid S * S \mid S/S \mid (S) \mid a$$

How many parse trees are there for each of the following:
(i) `a + a * a` (ii) `a + a * a / a` (iii) `a + a + a + a` (iv) `(a + (a + a)) + a`
Explain your answer in each case. Try to explain without drawing all the parse trees.

answer:

(i) There are two parse trees corresponding to the evaluation order `a + (a* a)` and `(a + a) * a`

(ii) The parse trees correspond to the evaluation order `a + (a * (a / a))`

```
a + ((a * a) / a)
(a + a) * (a / a)
(a + (a * a)) / a
((a + a) * a) / a
```

(iii) The parse trees correspond to the evaluation order `a + (a + (a + a))`

```
a + ((a + a) + a)
(a + a) + (a + a)
(a + (a + a)) + a
((a + a) + a) + a
```

(iv) There is only one parse tree. The parantheses in the expression allow only evaluation order

In (i), (ii) and (iii) parantheses are used only to show the evaluation order without drawing the parse trees.

4. Consider the following IMP program where `n` $\geq$ `0` initially. Find a loop invariant for the loop in the program. The stronger your loop invariant, the more useful it is (*e.g.* `true` is a loop invariant for all loops, but it is not very useful).

```
j := 0;
x := 1;
while j < n do
    j := j + 1; x := 2 * x
```

answer:

The loop invariant is

$0 \geq j \wedge j \leq n \wedge x = 2^j$

j varies between 0 and n while the loop executes. Also, in each iteration of the loop x is multipliled by 2. So, let x_{bef}, j_{bef} be the values of x,j at the beginning of an iteration. Let x_{aft}, j_{aft} be the values of x,j at the end of that iteration.

Then $x_{aft} = x_{bef} * 2$ and $j_{aft} = j_{bef} + 1$. Therefore if $x_{bef} = 2^{j_{bef}}$ then $x_{aft} = 2 * x_{bef} = 2 * 2^{j_{bef}} = 2^{j_{bef}+1}$.

5. Note that `n` is never modified in the program fragment of question A.4. Now by using the method proving via Hoare triples show that the property `x = 2^n` holds at the end of the while loop.

Answer:

```
let loop invariant = P, then we have

    {n >=0}
    j = 0;
    x = 1;

    {P}
    while j < n do
        j = j+1;
        x = 2*x;
    {j >= n ^ P}

    {x = 2^n}

before the while loop,
    P is true, because:

    - by sequence rule followed by assignment rule, we can deduce that

    {0 <= 0 ^ 0 <= n ^ 1 = 2^0}
    is true at the start of the program. The above is inturn implied by the
precon {n >= 0}

    so, by consequence rule, the hoare triple before the loop is true.

for the loop,

    P is loop invariant, because:

    - by the loop invariant rule, if the below is true, then P is
      invariant:

    {j < n ^ P } j = j + 1; x = 2*x; {P}

by sequence rule followed by assignment rule and using P, we deduce that

    0 <= (j+1) ^ (j+1) <= n ^ (2*x) = 2^(j+1) ------- Q

is true before the loop body.

    so if {j < n ^ P} implies the above, then we can use the consequence
rule to show that the loop preserve the P.

split Q into three parts:

    a: (0 <= (j+1))
        proof: Since  0 <= j,   a is true

    b: (j+1) <= n
        proof: Since j < n,  b is true
```

```
    c: (2*x) = 2^(j+1)
        proof: Since x = 2 ^ j, (2*x) => 2*2^j
                                       => 2^(j+1)
                so, c is true.

    Hence, the P is loop invariant.
at the end of the loop, if we can show {j>= n ^ P} implies post con, i.e.
{x = 2^n},we can rule of consequence to show the whole hoare triple is true.

since j >= n ^ j <= n,  we can deduce that j = n.

since x = 2^j, from the above, we can deduce that

    x = 2^n
```

Section B: Imperative and Object-oriented Programming
(3 + 6 + 2) = 11 marks

1. Consider the following class definitions taken from an object oriented programming language.

```
Class  Shape{
Protected :
  float   width;

Public :
  float  align;
Private:
    int  data;
}
```

```
Class  Text{
   protected:
        char  str;
        Int len;
}
```

```
Class Circle:Shape{
  private:
     float  rad
}
```

Among the fields width, align,data,str,len,rad which are visible to an instance of the Circle class and why ?

answer:

Circle object can access:

- width (protected, accessible in descedant class)
- align (public)
- rad (private, accessible in own class, i.e. Circle).

2. Explain what are the three numbers printed in case the parameter Y is transmitted by (i) name (ii) value (iii) reference (iv) value result.

```
program Main();
   var Y : integer;
   procedure Proc(X: integer);
        begin X := X + 1; write(X, Y) end
   begin
        Y := 1; Proc(Y); write(Y)
   end
end
```

answer:

- By name: By textual substitution, the code for Proc becomes
  begin Y := Y +1; write(Y, Y) end
  So it prints (2,2) in Proc. After returning it prints 2 since Y is then 2
- By value: Value of Y (global variable) is not changed. (2,1) is printed in Proc. After returning 1 is printed since Y is still 1.
- By reference: Updates to parameter X are immediately visible in global variable Y. Incrementing of X in Proc results in incrementing of Y. So, in Proc X = Y =2 when they are printed. After returning 2 is printed since Y is now 2.
- By Value result: 1 is copied to X. X is 2 when it is printed (after incrementing), but Y is still 1. So (2,1) is printed inside Proc. On return, the value of X (now 2) is copied to Y. So 2 is printed inside Main

3. Consider an array A

```
A : array[1..M,1..N] of integer
```

In class, we discussed the formula for computing the address of array element A[i,j] using the address of A[1,1]. This was done assuming that the array layout is in row-major order *i.e.* the first row appears first followed by the second row and so on. What will be the formula for the address of A[i,j] (in terms of the address of A[0,0]) if the array layout is in column major order. Explain with pictures. You may assume that any integer occupies 4 bytes of memory.

answer:

For row-major order we have

Addr(A[i,j]) = Addr(A[1,1]) + (i-1)*N*4 + (j-1)*4

This is because there are (i-1) full rows ahead of A[i,j]. furthermore (j-1) cells in the ith row are also ahead of A[i,j]. each row occupies N*4 bytes of memory (since N cells in each row).

By a similar argument in column major order

Addr(A[i,j]) = Addr(A[1,1]) + (j-1)*M*4 + (i-1)*4

This is because there are (j-1) full columns ahead of A[i,j]. furthermore (i-1) cells in the jth column are also ahead of A[i,j]. Each column occupies M*4 bytes of memory.