# Final Exam, CS6848, IIT Madras

06-May-2014

Read all the instructions and questions carefully. You can make any reasonable assumptions that you think are necessary; state them clearly. There are total six questions totaling 72 marks (including 12 bonus marks). Each twelve mark question will approximately take 30 minutes. For questions that have sub-parts, the division for the sub-parts are mentioned in square brackets.

Leave the first page empty. Start each question on a new page. Think about the question before you start writing and write briefly. **The answer for any question (including all the sub-parts) should NOT cross more than two pages.** If the answer is spanning more than two pages, we will ignore the spill-over text. If you scratch/cross some part of the answer, you can get compensation space from the next page.

1. [12] **Flow analysis I** For the following code generate the flow constraints using 0-CFA and solve the constraints.

```
class A1 implements I {
   I x = new A2();
   public I m(I y) {
    return x.m(y);
} } // end of A1
... new A2().m(new A1())
```

```
class A2 extends A1 {
   public I m(I y) {
      x.m(y).m(this);
} } // end of A2
```

2. [12] **Flow analysis II** We will assume a simplified procedural Java subset: a program consists of a set of classes, each class is a subtype of the Thread class (directly or indirectly) and consists of exactly one method (`run`), and at most one field. The `run` method takes no argument and consists of zero or more statements. Allowed statements: variable declaration, sequence, method call, object allocation, and print statements. Method overloading is not allowed. **Goal**: Write an algorithm that takes as input two statement labels and returns true if two statements may run in parallel. Apply your algorithm on the following code to confirm the results.
   **Hint**: Use a variation of flow analysis to identify which method may be called.

```
L1: class A1 extends Thread { L8: class A2 extends A1 {
L2:  A1 f2 = new A2();        L9:  A1 f1 = new A1();
L3:  void run() {             L10: void run() {
       Thread t1, t2;         L11:   f1.start().join();
L4:    t1 = f2.start();       L12:   S1; // serial code
L5:    t2 = new A3().start(); L13: } L14:} // end of A2
L6:    t1.join(); t2.join();  L15:class A3 extends A2 {
L7:  } } // end of A1         L16: void run() {
                              L17:   S2; // serial code.
  // main function            L18:   S3; // serial code.
  ... new A1().start().join();...  }} // end of A3
// Example May Happen in Parallel information:
// L17 may run in parallel with L4, L5, L11, L12, L17, L18
// L12 may run in parallel with L5, L17, L18
```

Notes:

- In Java `x.start()`, where the type of `x` is a subtype of `Thread` class, starts executing the `run` method of the thread in parallel with the current thread.
- `x.join()`, where the type of `x` is a subtype of `Thread` class, makes the current thread wait for thread `x` to finish executing (its `run` method).

3. [12] **Language extension** Consider an extension to C (with no pointer arithmetic), where the arguments may be passed by name (aka. Lazy evaluation) – let us call this CLazy. Give a mechanism to translate a program in such a language to normal C. The translation should ensure that if the input CLazy program terminates with eager evaluation and outputs a value $v$, then the value output by the translated program would match $v$.

```
int foo() {
        int x = 5;
        printf ("%d ",bar (5, x + fbar1(5, x)));
        printf ("%d ",bar (inf(), x + fbar1(inf(), x)));
}
int bar(int x, int y) {
        printf ("%d ",y); return y;
}
int fbar1(int x, int y) {
  if (y < 0) return x; return y - 1;
}
int inf() { while (1); return 0;}
```

The above program should print : `9 9 9 9`.

4. [4 + 4 + 4] **Axiomatic semantics I**

i) A CS6848 student when asked to derive the universal pre and post conditions, answered that the universal pre-condition is `true` and the universal post-condition is `false`. Is the answer correct? Prove it.

Prove that the following code performs the division of positive numbers. After the end Y has the quotient and X has the reminder. ii) State the post-condition and the pre-condition. iii) Derive the Hoare-triples at each program point to prove the above stated code property.

```
X ::= a;
Y ::= 0;
WHILE b <= X DO
    X ::= X - b;
    Y ::= Y + 1
END;
```

5. [12] **Axiomatic semantics II** Prove that the following code stores the value of `x - y` in the variable `X`.

```
X ::= x;
Y ::= y;
WHILE Y ≠ 0  DO
    X ::= X - 1;
    Y ::= Y - 1
END;
```

6. [6 + 6] Answer any of the two questions.

   (a) Communicating Sequential Processes: Using primitives similar to that used in the CSP paper, write a program to simulate Conway's game of life. Take a grid of $N \times N$ cells, each of which is either *alive* or *dead*. In each generation, each cell interacts with its neighbor to find their state and changes its own state based on the following rules:

      i. (no friends) A live cell with one or fewer live neighbors dies.

      ii. (healthy) A live cell with two or more live neighbors continues to live to the next generation.

      iii. (population explosion) A live cell with more than three live neighbor dies.

      iv. (resurrection) A dead cell with exactly three live neighbors becomes live.

   (b) Eiffel: Give a scheme to transform Eiffel code to C. Focus mainly on the following three constructs: `require`, `ensure`, and `retry`.

   (c) Proof Carrying Code: Compare and contrast the approach of PCC Vs Typed Assembly language.

   (d) Briefly describe (with an example) how flow analysis can help in improving the precision of automatic predicate abstraction.

   (e) Axiomatic semantics: Derive a loop invariant for the following code; our goal: to prove that the code computes the factorial function.

```
X ::= x
Y ::= 1
WHILE X <> 0 DO
     Y ::= Y * X;
     X ::= X - 1
END
```