CS6235 Final Exam: May 11 2023

Maximum marks = 50, Time: 2.00 hrs, Closed Book, Closed Neighbor

Name: ______________________________ Roll: ____________

- **Write your roll number on each of the answer sheet.**
- The marks of each question is specified next to the question.
- Advise: start each question on a new page.
- True or False questions: Each incorrect True/False answer will lead to a deduction of 0.5 mark.

---

1. **Escape Analysis** [5]
   - Write a sample Java code, where the flow-sensitive and flow-insensitive thread escape-analysis will give different results. [2]
   - For your written code, mention the thread escape-analysis results for both the flow-sensitive and flow-insensitive dimensions. [2]
   - [True/False] In the absence of thread escape-analysis we can use strong updates on the stores in parallel Java applications. [1]

2. **Deadlock detection** [5] Consider the deadlock detection scheme discussed in the class using resource allocation graph.
   - Write an example Java code where the deadlock detection algorithm will give a false positive (that is, the algorithm reports the presence of deadlock, but the deadlock will not occur in practice). [2.5]
   - Show the resource allocation graph at the points where the deadlock is reported. [1.5]
   - [True/False] The deadlock detection algorithm has no false negatives. [1]

3. **Memory consistency model.** [5]
   - Give a memory trace that is not consistent as per sequential consistency, but is consistent as per weak ordering. [2]
   - Give a memory trace that is not consistent as per TSO, but is consistent as per PSO. [2]
   - [True/False] If a trace satisfies TSO, it is guaranteed to satisfy Weak ordering.

4. **Memory consistency model: Programmer's view** [5]
   - Give an example that shows that sequential consistency is not easy to use from the programmer's point of view. [2]
   - For the above example you have written, mention the best consistency (along with an explanation) from the programmer's point of view. [2]
   - [True/False] A DRF-0 program may not satisfy SC. [1]

5. **Data-Race Detection** [5]
   - Write a Java program and its trace which shows HB based race, but no CP based race. Show the HB and CP based orderings and give the reasoning for race. [4]
   - [True/False] There can be a trace which leads to a CP based race, but not an HB based race. [1]

6. **Serial program analysis-I.** [5]
   - Write a program that shows how a context-sensitive points-to analysis is more precise than context-insensitive points-to analysis. [2]
   - Write a program that show how a inter-procedural points-to analysis is more precise than intra-procedural points-to analysis. [2]
   - [True/False] In the most conservative must points-to analysis, each variable points to the empty set. [1]

7. **Serial program analysis-II.** [5]
   - Answer the following question in the context of intra-procedural points-to/alias analysis of Java programs discussed in the class [1 × 4]:
     (i) What is the size of the lattice?
     (ii) What is the abstract stack (ρ) initialized to at the beginning of a function?
     (iii) Between the abstract stack (ρ) and abstract heap (σ), in general, which one has more number of elements?
     (iv) During the analysis of a function call statement, whose contents (ρ or σ) are guaranteed to not change, if any?
   - If in a function, no variable is defined more than once, flow sensitivity does not improve the precision. [1]

8. **MHP Analysis** [5].
   - Recall the incremental MHP analysis studied in the class. Give an example parallel program written in a language with `finish`-`async` constructs such that if we re-introduce the `async` constructs, followed by the `finish` constructs, then the computed MHP information would be imprecise. [2]
   - If we re-introduce the `async` constructs, followed by the `finish` constructs, then the computed MHP information can be unsound. If yes, give an example program. Else prove otherwise. [2]
   - [True/False] Flow insensitive points-to analysis requires MHP analysis. [1]

9. **Intermediate Representation** [5].
   - Write a C function with more than 2 statements, such that the number of basic-blocks (excluding the start and end basic blocks) is equal to the number of "semicolon" terminated statements in the function. [2]
   - Write three-address code for the following C code.

```
// Assume that the array A is declared as
// int A[4][5];
i = 0;
j = 0;
do
 do
   sum += A[i][j];
   j++;
  while (j < 5);
 i++;
 while (i < 4);
```

   - [True/False] The number of nodes in the PEG can be greater than the number of statements of the program.

10. **Liveness analysis**
   - Write a program, where using the equation $in[n] = use[n] \cup out[n] - def[n]$ will lead to incorrect results. [2]
   - Write a program, with more than 2 variables, such that each variable is live only at one statement. [2]
   - [True/False] If a variable is live-out at a node, it is live-in at each of its successors. [1]