CSL862 Minor1 Exam
Advanced Topics in Software Systems
Sem I, 2013-14
August 29, 2013

Answer all 5 questions Max. Marks: 44

1. Answer **True/False**. Give reasons. No marks if no reason (or incorrect reason) given.
    a. Without using the happens-before graph, it is possible for the CHESS algorithm to never finish on a program (assume that the program is terminating and CHESS algorithm finishes when using the happens-before graph). Explain with examples if needed. [4]

    b. Consider a program using Dthreads: Adding an extra non-synchronization instruction to the middle of this program can cause the interleaving behaviour of the program to change. Explain with example, if needed. [4]

2. Consider Derandomized-PCT where a deterministic and systematic exploration of the schedules is performed in increasing order of bug depth. i.e., first all schedules to uncover depth-1 bugs are explored exhaustively, then all schedules to uncover depth-2 bugs are explored exhaustively, then all schedules to uncover depth-3 bugs are explored exhaustively, and so on . . .

a. Assume a program with n threads and a total of k instructions. After how many schedules (in order notation) are we guaranteed to find a bug of depth d? [4]

b. Give an example of a program with a bug, such that the bug is of depth 2 and requires at least 2 pre-emptive context switches to be exposed (in CHESS). The program need not do anything useful. You can use the ASSERT statement in your program, where an assertion failure indicates a bug. [8]

c. Give an example of a program with a bug, which will be found sooner with CHESS than with Derandomized-PCT? The program need not do anything useful. You can use the ASSERT statement in your program, where an assertion failure indicates a bug. [8]

3. PRES works by recording certain sources of non-determinism (e.g., result of synchronization operations) and ignoring others (e.g., data races). During replay, it explores the search space of the unrecorded non-determinism to try and reproduce the bug that caused a failure during production run. Answer the following questions

a. During replay, the “feedback generator” generates feedback for future replays. What kind of feedback is generated, and how is it used? [4]

b. While searching for the bug during replay, does it make sense to use context-bounding (from CHESS)? If so, how and why? If not, why not? [6]

4. Consider the following program:

```
int x = 1, y = 2;
thread_fork(child_thread);
```

parent thread	child thread
x = y;	y = x;

```
thread_join(child_thread);
printf("x=%d, y=%d\n", x, y);
```

Assume that the statements "x = y" (or "y = x") **atomically** move the contents from memory

location ‘y’ to memory location ‘x’ (or from memory location ‘x’ to memory location ‘y’ respectively).

What are the possible final contents of x and y with

a. pthreads?
b. dthreads?

Explain. [4]

5. Translation validation: What is a simulation relation? How is it verified? [2]

Homework 2

To be done individually without discussion with anybody else (except instructors and TAs).

Due on September 5, 2013

Consider the following two functions foo() and bar().

```
int a[];  /* global array */

void foo(int n) {
   int i = 0;
   while ((2 * i + 1) < n) {
      a[i * 2] = (3 * i) + 4;
      a[i * 2 + 1] = i + 1;
      i++;
   }
   return;
}
```

```
void bar(int n) {
    int j = 0;
    int sum_even = 4;
    int sum_odd = 1;
    for (j = 0; j < n; j++) {
        if ((j % 2) == 0) {
            a[j] = sum_even;
            sum_even += 3;
        } else {
            a[j] = sum_odd;
            sum_odd += 1;
        }
    }
    return;
}
```

Assume that n is even. Also, assume that the values n, sum_even, sum_odd, etc. do not overflow the integer data type. Using mathematical induction, prove that these programs are equivalent. Will the approach in the translation validation paper be able to infer the simulation relations in this case? If so, what will be the simulation relations? If not, why not?