CS510 Assignement #4 (Due 4/28 before class)

April 14, 2015

1 Symbolic Model Checking (30p)

For this problem, you will need to convert a program into a set of constraints. The constraints should be such that every satisfying assignment corresponds to an execution of the program that violates an assertion. The program is given below:

```
void main (void)
{
  int    i = input();
  char * buf = input2();
  char   next = 0;

  if (buf [i] == '\0')
    {
      int start = 0;
      while (start < i)
        {
          buf [start] = '.';
          start++;
          next = buf [start];
        }
    }
}
```

(a) Show the result of unrolling the loop twice (2) and adding an unwinding assertion.

(b) Convert the loop-free program (with assertion) from part (a) into Single Static Assignment (SSA) form.

(c) Generate a constraint system C from the SSA program in part (a) such that C is satisfiable if and only if the unwinding assertion can be violated. Note that you don't need to turn it into boolean formula.

(d) Is the constraint system C from part (c) satisfiable? If yes, show a satisfying assignment.

2 Predicate Abstraction (30p)

In order to mitigate state explosion in explicit state model checking, predicate abstraction is often used to reduce state space. Counter-example guided refinement may be needed during the process.

```
void main (void)
{
   int a, b;
   a = 2;
   b = 5;
   if (a > b) {
      a--;
   } else {
      a+=4;
   }

   assert(a>b);
}
```

(a) Starting with predicate `a>b`, apply predicate abstraction to the above program.

(b) Perform explicit state model checking on the abstract program, present your execution tree and the counter example, if there is one.

(c) If there is a counter example in (b), test if it is a counter example in the original program.

(d) If the counter example is bogus, refine your abstraction so that either you find a real counter example or show the correctness of the program.

3 Concolic Execution (25p)

```
input (x,y,z);
input (A[]);
a=2*x;
b=y*y;
c=x+1;
if (a>b) {
  c=c+1;
  if (b>0)
     c=c+1;
  else
     c=c+2;
}
if (A[z]>c) {
  print ("case 1");
```

```
} else {
  print ("case 2");
}
```

Apply concolic execution to the above program. The initial input is x=3, y=4, z=1, A[]={3,2,1}. Show the path constraints generated at each iteration of the algorithm and the generated input. What is the final statement coverage? Assume the solver can not solve non-linear or array index related constraints.

4 Concurrency (15p)

Thread T1

```
0  r=input();
1  B=1000;
2  spawn(T2);
3  ...
4  acquire (L);
5  t=input ();
6  B=B+t;
7  release (L)
8  ...
9  join (T2);
```

Thread T2

```
10  acquire (L);
11  y=input ();
12  B=B-y;
13  release (L)
      ...
14  if (r>0) {
15     acquire (L);
16     B=B+y*r;
17     release (L);
18  }
```

One way to decide if a concurrent program has a bug is to run the program on the same input many times and observe if the same output is always produced. If not, the program is faulty as it does not produce stable output. Is the above program faulty? If so, present the input and the two schedules that produce different outputs.

For input r=0, t=10, y=20, how many races (pairs of accesses) are reported following the lockset algorithm? How many following the happens-before algorithm.