CS615 Endsem Exam (Autumn 2017)

Max marks: 60 **Time: 180 mins**

- *Be brief, complete and stick to what has been asked.*
- *Unless asked for explicitly, you can cite results/proofs covered in class.*
- ***If you need to make any assumptions, state them clearly.***
- ***Read the question paper carefully before answering questions.***
- ***Do not copy solutions from others. Penalty for offenders: FR grade.***

1. *[5 + 5 + 5 + 10 marks]*

   Consider the following program in a C-like language, where all variables are of type `int`:

```
L1:  x = y;
L2:  while (x < 1000) {
L3:    y = y + 1000;
L4:    x = 1000 - y;
L5:  }
L6:  assert (!((0 <= y) && (y < 1000)));
```

   Note that `int` variables can assume positive, negative and zero values.

   A student wishes to use predicate abstraction to determine if the above program, when executed in a state satisfying the pre-condition True can violate the assertion. She starts off with an initial choice of predicates $\mathcal{P} = \{p_1 \equiv (x < 1000), p_2 \equiv (y < 1000), p_3 \equiv (y \geq 0)\}$.

   (a) Fill in the blank below with the most precise expression in terms of `p1, p2, p3, *` to complete the Boolean program corresponding to the above choice of predicates.

```
L1:  p1 = p2;
L2a: while (*) { // non-deterministic choice
L2b:  assume(p1);
L3:   (p2, p3) = ( _______________ , (!p2 || p3) ? 1 : * );
L4:   p1 = !p2 ? 1: !p3 ? 0 : *;
L5:  }
L2c: assume(!p1);
L6: assert(!(p2 && p3));
```

   (b) Show that `L1, L2a, L2b, L3, L4, L5, L2a, L2c, L6` is an abstract counter-example trace, i.e, a trace of the abstract program that violates the assertion. In other words, you have to provide values of p_1, p_2, p_3 at the start of the Boolean program that causes the instructions in the above trace to be executed and the assertion (in the Boolean program) to be violated.

(c) Construct the trace formula (from the original program statements) corresponding to the above abstract counter-example trace.

(d) Is the trace formula constructed above satisfiable?

If so, solve the trace formula to obtain a value of y at the start of the original program that leads to a violation of the assertion after iterating through the while loop (in the original program) once.

Otherwise, use Craig interpolation to identify the *smallest* set of additional predicates that need to be added at each location of the original program to ensure that the resulting predicate abstraction excludes this counterexample trace.

2. *[10 + 10 marks]* Consider the following program in a C-like language:

```
L1:  x = y*10;
L2:  while (x < 1000) {
L3:    if (y < 2000) {
L4:      x = 1000 - y;
L5:    }
L6:    else {
L7:      y = y*5;
L8:    }
L9:    if (x < 1000) {
L10:     y = y*2;
L11:   }
L12:   else {
L13      x = 2000 - y;
L14:    }
L15: } // end of while loop
L16: assert (phi(x, y)); // phi is a predicate on x, y
```

(a) We wish to use bounded assertion checking to determine if the assertion at `L16` can be violated starting from the pre-condition $\psi(x, y)$ using *at most* one iteration of the `while` loop. You may assume that $\psi(x, y)$ is a predicate on x and y.

Write a quantifier-free formula with φ and ψ that evaluates to `True` iff the assertion can be violated in at most one iteration of the loop.

Your formula must not involve any uninterpreted predicates other than φ and ψ, and must be linear in the size of the program (i.e. it *must not* be based on enumerating potentially exponentially many paths in the program.)

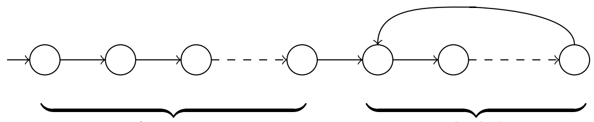
(b) Write a *predicate-minimal* formula in Horn Logic, where $\varphi(x, y)$ and $\psi(x, y)$ are treated as predicates, such that

- The formula is satisfiable iff the assertion $\varphi(x, y)$ is always true at `L16` when the program is executed starting from the pre-condition $\psi(x, y)$.
- The formula uses as few uninterpreted predicates other than $\varphi(x, y)$ and $\psi(x, y)$ as possible.

Solutions that use more than the minimum number of predicates may lose marks in the evaluation.

[Hint: You can try to minimize the number of predicates by partially solving the Horn formula]

3. *[5 + 10 marks]* A *lasso* is a non-NULL terminated singly linked list of the shape shown below:



Note that both the *noose* and the *handle* of the lasso must have at least one node. Thus, the smallest lasso must have at least two nodes.

(a) Give an inductive definition of $lasso(v, n, m)$ in separation logic that evaluates to True iff the following conditions are satisfied:

- The heap has a single lasso and the first node in the handle of the lasso is v.
- The lasso has a handle of n nodes and a noose of m nodes.

You may assume that each node in a lasso has a field named `n` that contains a pointer to the next node in the lasso.

(b) Consider the program below in a C-like language:

```
void ShrinkNoose(lassoPtr x, int n, int m) {
    if (n > 1)
      ShrinkNoose(x->n, n-1, m);
    else { // n == 1
      nFirstNode = x->n;
      if (nFirstNode->n == nFirstNode)
         return;
      else
         ShrinkShortNoose(x, m);
    return;
```

Assume that the pre-condition for `ShrinkNoose` is $lasso(x, n, m)$. Furthermore, assume that `ShrinkShortNoose` satisfies the following specification:

$$\{lasso(x, 1, m) \wedge (m > 1)\}\ \ \texttt{ShrinkShortNoose(x,m)}\ \ \{lasso(x, 1, \max(1, m-1))\}$$

Use the Frame Rule and induction in separation logic to prove the following Hoare triple:

$$\{lasso(x, n, m)\}\ \ \texttt{ShrinkNoose(x,n,m)}\ \ \{lasso(x, n, \max(1, m-1))\}$$