CS615 Midsem Exam (Autumn 2015)

Max marks: 75 **Time: 120 mins**

- *Be brief, complete and stick to what has been asked.*
- *Unless asked for explicitly, you may cite results/proofs covered in class without reproducing them.*
- ***If you need to make any assumptions, state them clearly.***
- ***Do not copy solutions from others. Penalty for offenders: FR grade.***

1. Consider the following program in a C-like language, in which conditional assignment statements are used. Thus, a statement like `b = !h ?  a+b:  a` is semantically equivalent to `if (!h) b := a+b; else b := a;`.

```
     int a, b; bool h;
L1:  while (a != b) do {
L2:    b := !h ? a+b : a;
L3:    h := (a != b) ? true : h;
L4:  }
L5:  assert (h);
```

   A student wants to analyze the above program using predicate abstraction (or equivalently, Boolean programs) to determine if the assertion at line L5 can be violated starting from a pre-condition (to be specified). The student has decided that she will use the set of predicates $P = \{p_1, p_2\}$, where p_1 represents $(a = b)$, and p_2 represents $(h = \mathsf{true})$.

   (a) *[10 marks]* Construct as precise a Boolean program $\mathcal{BP}$ as you can using the set of predicates P. To score marks, you must make your Boolean program precise enough so that we can correctly determine whether the assertion at line L5 of the original program holds for the pre-conditions $\{h = \mathsf{true}\}$ and $\{h = \mathsf{false}\}$.

   (b) *[10 marks]* Construct the finite state transition diagram corresponding to $\mathcal{BP}$ obtained above.

   (c) *[5+5 marks]* Using the finite state transition diagram obtained, show the following:

      i. The assertion at line L5 cannot be violated starting from the pre-condition $\{h = \mathsf{true}\}$.

      ii. The shortest counterexample trace violating the assertion at line L5 starting from the pre-condition $\{h = \mathsf{false}\}$ is not a spurious counterexample trace.

2. We have studied in class that given an abstract domain $(\mathcal{A}, \sqsubseteq, \sqcup, \sqcap, \top, \bot, \nabla)$, the widening operator $\nabla : \mathcal{A} \times \mathcal{A} \rightarrow \mathcal{A}$ satisfies the following properties:

- For every $a, b \in \mathcal{A}$, we have $a \sqsubseteq a \nabla b$ and $b \sqsubseteq a \nabla b$.
- For every non-decreasing sequence of elements $x_0 \sqsubseteq x_1 \sqsubseteq \ldots$ in $\mathcal{A}$, the following sequence of a_i's stabilizes (ceases to change) after finitely many steps:
  - $a_0 = x_0$
  - $a_{i+1} = a_i \nabla x_{i+1}$

(a) *[10 marks]* Show that the above definition doesn't guarantee monotonicity of the widen operator. Specifically, give an example of an abstract domain and definition of ∇ that satisfies all the properties given above, and yet there exist elements $a, b, c \in \mathcal{A}$ such that $b \sqsubseteq c$ and $(b \nabla a) \not\sqsubseteq (c \nabla a)$.
[Hint: Think of the different abstract domains studied in class.]

(b) *[10 marks]* Consider the abstract domain of *conditional convex polyhedra* used in Quiz 2. In other words, every abstract element is a triple (C, P_1, P_2), where C is a boolean condition (over boolean and numerical variables in the program) and P_1 and P_2 are convex polyhedra (over numerical variables in the program). As discussed in Quiz 2, the triple (C, P_1, P_2) represents "if (C) then P_1 else P_2". More formally, $\gamma((C, P_1, P_2)) = \{s : s \models (C \wedge P_1) \vee (\neg C \wedge P_2)$. We will say that $(C, P_1, P_2) \sqsubseteq (C', P_1', P_2')$ iff $\gamma(C, P_1, P_2) \subseteq \gamma(C', P_1', P_2')$.
Let ∇_{poly} denote a widen operator in the domain of convex polyhedra. Using ∇_{poly}, define a suitable widen operator in the domain of conditional convex polyhedra. You must show that all properties required of a widen operator are satisfied by your definition.

3. *[5×5 marks]* In this question, we'll try to compute the strongest abstract post-conditions (in the interval abstract domain) of various C-like assignment statements. Assume that all variables of interest are of type `int` and the domain of interest is that of intervals. Specifically, we have an *open interval* (l_x, u_x) for every `int` variable `x` in the program, where $l_x \in \mathbf{N} \cup \{-\infty\}$ and $u_x \in \mathbf{N} \cup \{+\infty\}$. Every program statement computes (potentially new) values of the bounds l_x and u_x for every program variable `x`. The concretization of the interval (l_x, u_x) gives all concrete states in which $l_x < x < u_x$ (note the strict inequalities).

   In each of the following sub-problems, you must indicate how the new values of l_x and u_x should be computed to obtain as tight an interval abstraction of the post-condition of each statement, as possible. The expressions for l_x and u_x can, of course, be C-style expressions in terms of the (lower and upper) bounds for variables prior to the execution of the statement. A solved example is given below.

   (a) **Solved example:** `x := x + y;`
   Answer: l_x `:=` l_x `+` l_y `+ 1;` u_x `:=` u_x `+` u_y `- 1;`

   (b) `x := x*y + z;`

   (c) `x := x/y;`

   (d) `x := (y > 5) ?  x :  y;`

   (e) `x := x*x + y*y;`

   (f) `x := (x != 0) ?  1:  0;`