CS615 Midsem Exam (Autumn 2016)

Max marks: 50 **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. Let $\mathcal{A} = (A, \sqsubseteq, \sqcup, \sqcap, \nabla, \top, \bot)$ be an abstract lattice, and let P be the program fragment `if` (B) `then` P_1 `else` P_2.

   We will use α and γ to denote the usual abstraction and concretization functions, and assume that (α, γ) forms a Galois connection between the lattice of sets-of-program-states and $\mathcal{A}$.

   Suppose the concrete state transformers for P_1 and P_2 (i.e. F_1^C and F_2^C) and the corresponding abstract state transformers (i.e. F_1^A and F_2^A) satisfy $\forall a \in A$, $\alpha(F_i^C(\gamma(a)) = F_i^A(a)$, for $i \in \{1, 2\}$.

   (a) *[10 marks]* Using notation defined in class, let $F_P^A(a)$ be defined as $F_1^A(a \sqcap \alpha(B)) \ \sqcup \ F_2^A(a \sqcap \alpha(\neg B))$, for all $a \in A$.
   Does it follow that $\alpha(F_P^C(\gamma(a)) = F_P^A(a)$ holds for all $a \in A$, where F_P^C denotes the concrete state transformer of P?
   If your answer is "Yes", provide a complete proof. Otherwise, provide a a specific program $P =$ `if` (B) `then` P_1 `else` P_2 and a specific abstract domain $\mathcal{A}$, and show that this serves as a counterexample.

   (b) *[10 marks]* A student has defined a new abstract state transformer FF_P^A, for the program fragment P, as follows.

   $$FF_P^A(a) = F_1^A(\alpha(\gamma(a) \cap B)) \ \sqcup \ F_2^A(\alpha(\gamma(a) \cap (\neg B)))$$

   The student claims that her state transformer is sound and provably no worse than F_P^A defined in the previous sub-question. In other words, she claims that $\forall a \in A$, $\alpha(F_P^C(\gamma(a))) \sqsubseteq FF_P^A(a) \sqsubseteq F_P^A(a)$.
   If you think the claim is correct, provide a complete proof. Otherwise, provide a a specific program $P =$ `if` (B) `then` P_1 `else` P_2 and a specific abstract domain $\mathcal{A}$, and show that this serves as a counterexample.

   (c) *[10 marks]* Suppose $\mathcal{A}$ is the interval abstract domain studied in class. Let FF_P^A and F_P^A denote the abstract state transformers introduced in the previous sub-problems.
   Give an example of a program $P =$ `if` (B) `then` P_1 `else` P_2 in the simple programming language discussed in class, and an abstract state (of intervals) a such that the conditions on F_1^A and F_2^A given above hold, and in addition, each of the following conditions hold:

- $F_P^A(a) \neq FF_P^A(a)$
- $FF_P^A(a) \neq \alpha(F_P^C(\gamma(a)))$
- $F_P^A(a) \neq \alpha(F_P^C(\gamma(a)))$

2. Consider the following program in the language studied in class.

```
     int a, b;
L0:
L1:  while (a > b) do {
L2:    b := b + 1;
L3:    a := a - b;
L4:  }
```

You are told that the pre-condition at location L0 is
$(100 < b < 200) \wedge ((0 < a < 1000) \vee (b < a < b + 1000))$.

We wish to compute as strong a loop invariant as we can at location L1 using the abstract domain $\mathcal{D}$ of Difference Bound Matrices (DBMs). Towards this end, you are required to answer the following questions.

(a) *[5 marks]* Find the best abstract state transformer $F^A : \mathcal{D} \to \mathcal{D}$ of the loop body (statement at L2 followed by that at L3) in the DBM domain. Your abstract state transformer should take a DBM as argument and return another DBM.

(b) *[5 marks]* Give a monotone map $h : \mathcal{D} \to \mathcal{D}$ from the abstract lattice of DBMs back to itself such that *any post-fix point* of h other than $\top$ gives a non-trivial abstract loop invariant of the above program at location L1. Your monotone map should take a DBM as argument and return another DBM.

(c) *[10 marks]* Compute the best possible abstract loop invariant at L1 using the monotone map h obtained in the previous sub-question. You can defer the use of the widen operator (using *lub* instead) for two iterations when computing the abstract loop invariant. You must clearly show all steps in your calculation.