CS615 Quiz 2

Max marks: 20 **Time: 30 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 Problem

Consider the following program `P` in which all variables are of type `int` or `bool`.

```
     int x, y, z; bool b;
L0:  while (z > 0) do {
L1:    if (x = y) then {
L2:       z := 2*x;
L3:       b := true;
L4:       x := x - 1;
L5:    }
L6:    else {
L7:       z := x + y;
L7:       b := false;
L8:    }
L9:    z := z - 1;
L10: }
```

Assume that the pre-condition for this program is $\{x > 0 \wedge y > 0 \wedge z > 0\}$.
We wish to use a new abstract domain to analyze this program. In this domain, each abstract element is a triple (C, P_1, P_2), where C is a boolean condition and P_1 and P_2 are convex polyhedra over the numeric variables in the program. As an example, the following is a (rather arbitrarily chosen) abstract element describing a set of concrete states of the above program: $((b \vee (x = y)), (z = 2x + 1) \wedge (y > 0), (z = x + y) \wedge (y \geq 0))$, where $b \vee (x = y)$ is the boolean condition C, and the polyhedra P_1 and P_2 are $(z = 2x + 1) \wedge (y > 0)$ and $(z = x + y) \wedge (y \geq 0)$ respectively.
Intuitively, an abstract element (C, P_1, P_2) represents the set of states "if (C) then P_1 else P_2". More formally, the concretization function γ is given by:

$$\gamma((C, P_1, P_2)) = \{s : s \models ((C \wedge P_1) \vee (\neg C \wedge P_2))\},$$

where s represents a concrete state. You may choose to use any appropriate abstraction function α that, along with γ, satisfies the definition of a Galois connection.
Notice that the above abstract domain strictly generalizes the polyhedra domain, since any polyhedron P can always be expressed in the new domain as (C, P, P), where C is an arbitrary boolean condition.

Question [20 marks]: Compute as strong an abstract loop invariant (in the new abstract domain described above) at `L0` as you can using techniques studied in class. You can either guess the abstract loop invariant at `L0` and then prove using Hoare logic rules that it is indeed a loop invariant, or derive the abstract loop invariant using abstract interpretation techniques studied in class.

2 Solution

***Important note:* The following is not the only way to solve the problem. It is perfectly possible that you have come up with a completely different, yet correct, way to solve the problem. Please let the instructor know if you have solved the problem in a different way.**

- We will use the technique of abstract interpretation to compute the abstract loop invariant.
- Recall that if the initial abstract state (when we reach `L0` for the first time) is a_0, then the abstract loop invariant at `L0` can be computed as the limit of the following sequence of a_i's:
- $a_0, a_1 = a_0 \nabla x_1, a_2 = a_1 \nabla x_2, a_3 = a_2 \nabla x_3 \ldots$, where
  $x_0 = a_0, x_1 = G(a_0), x_2 = G(a_1), x_3 = G(a_2), \ldots$ and $G(a) = a_0 \sqcup F(a \sqcap \alpha(B))$.
- In the above sequence, we could use $\sqcup$ for calculating the first few (fixed number of) a_i's, and then fall back on ∇. As discussed in class, this doesn't affect the property of the sequence becoming stable after a finite number of indices (or steps). You might also recall from the discussion in class that F is the abstract state transformer (or an overapproximation of it) corresponding to the body of the loop, and B is the loop condition.
- Now, let's compute the above sequence of a_i's for the specific problem in this quiz. You'll notice that we need to define the $\sqcup$ and $\sqcap$ operators, also the transformer F, but let's postpone these definitions to the point where we really need them.
- Since the pre-condition is $(x > 0) \wedge (y > 0) \wedge (z > 0)$, we could use $a_0 = (C, P, P)$, where C is any arbitrary boolean condition and P is the polyhedron represented by $(x > 0) \wedge (y > 0) \wedge (z > 0)$. Note that this choice of a_0 does not require us to freeze C at this point. In fact, we can choose C to be something convenient for us, as we proceed further.
- Therefore $x_0 = a_0 = (C, P, P)$. To compute x_1, we need to evaluate

$$G(a_0) = a_0 \sqcup F(a_0 \sqcap \alpha(z > 0)).$$

- What is $\alpha(z > 0)$? A minute's thinking will convince you that $\gamma((C', z > 0, z > 0))$ represents the set of concrete states $\{s : s \models z > 0\}$, for any arbitrary boolean condition C'. Therefore, we can use $(C', z > 0, z > 0)$ for $\alpha(z > 0)$, where C' is any boolean condition that we are free to choose (and we should choose this in a way later that simplifies our calculations).

- Assuming $a = (C, P_1, P_2)$ in the general case, what is $a \sqcap \alpha(z > 0)$? By definition, $\gamma((C, P_1 \wedge (z > 0), P_2 \wedge (z > 0)))$ represents the set of concrete states $\{s : s \models (C \wedge P_1 \wedge (z > 0)) \vee (\neg C \wedge P_2 \wedge (z > 0))\}$. It is easy to see that this is nothing but $\{s : s \models (C \wedge P_1) \vee (\neg C \wedge P_2)\} \cap \{s' : s' \models z > 0\}$. In other words, $\gamma((C, P_1 \wedge (z > 0), P_2 \wedge (z > 0))) = \gamma(C, P_1, P_2) \cap \{s' : s' \models z > 0\}$. Therefore, we will consider $a \sqcap \alpha(z > 0)$ to be $(C, P_1 \wedge (z > 0), P_2 \wedge (z > 0))$. Note that in this case we could use $z > 0$ to constrain each of P_1 and P_2 because $z > 0$ admits a polyhedral representation.

- *Notice also that we got away without defining $\sqcap$ in its full generality. We only defined it for the case we needed.*

- If $a = (C, P_1, P_2)$, we now need to compute $F(a \sqcap \alpha(z > 0))$, or equivalently $F((C, P_1', P_2'),$ where P_1' is $P_1 \wedge (z > 0)$ and P_2' is $P_2 \wedge (z > 0)))$. Let $\mathsf{SP}^a((C, P_1', P_2'), \texttt{Prog})$ denote the strongest abstract post-condition of (C, P_1', P_2') with respect to the program fragment `Prog`. Recall from our discussion in class that if SP^c denotes the strongest concrete post-condition operator, then we must have $\alpha \circ \mathsf{SP}^c \circ \gamma \ \sqsubseteq \mathsf{SP}^a$.

- In our case, the body of the while loop has two branches of an if-then-else statement. For the "then" branch, the abstract post-condition at `L5` is $\mathsf{SP}^a((C, P_1', P_2') \sqcap \alpha(x = y), \texttt{Prog}_{then})$, where $\texttt{Prog}_{then}$ is `z := 2*x; b := true; x := x-1;`. Using the same reasoning as above for computing $\sqcap$, we can write $(C, P_1', P_2') \sqcap \alpha(x = y)$ as $(C, P_1' \wedge (x = y), P_2' \wedge (x = y))$. Further, recalling that SP^a is (an overapproximation) of $\alpha \circ \mathsf{SP}^c \circ \gamma$, the desired result is the abstraction of $\{s : s \models \mathsf{SP}^c((x = y) \wedge ((C \wedge P_1') \vee (\neg C \wedge P_2')), \texttt{Prog}_{then})\}$. Let R_1 denote the polyhedral abstraction (over x, y and z) of $\mathsf{SP}^c((x = y) \wedge C \wedge P_1', \texttt{Prog}_{then})$, and let R_2 denote the polyhedral abstraction (over x, y and z) of $\mathsf{SP}^c((x = y) \wedge \neg C \wedge P_2', \texttt{Prog}_{then})$. Therefore, $\mathsf{SP}^c((x = y) \wedge ((C \wedge P_1') \vee (\neg C \wedge P_2')), \texttt{Prog}_{then})\}$ is abstracted by $\mathsf{CH}(R_1, R_2)$, where $\mathsf{CH}(\cdot, \cdot)$ represents the convex hull operator for polyhedra.

- Using a similar reasoning for the "else" branch, the abstract post-condition at `L8` is $\mathsf{SP}^a((C, P_1', P_2') \sqcap \alpha(x \neq y), \texttt{Prog}_{else})$, where $\texttt{Prog}_{else}$ is `z := x + y; b := false;`. Unfortunately, the polyhedral abstraction of $x \neq y$ is $\top_{poly}$ (top element in the polyhedra domain), since it is not possible to express $x \neq y$ as a conjunction of linear inequalities. Therefore, for the "else" branch, we need to compute $\mathsf{SP}^a((C, P_1', P_2'), \texttt{Prog}_{else})$. The corresponding concrete post-condition is $\{s : s \models \mathsf{SP}^c((C \wedge P_1') \vee (\neg C \wedge P_2'), \texttt{Prog}_{else})\}$. Let R_3 denote the polyhedral abstraction (over x, y and z) of $\mathsf{SP}^c(C \wedge P_1', \texttt{Prog}_{else})$, and let R_4 denote the polyhedral abstraction (over x, y and z) of $\mathsf{SP}^c(\neg C \wedge P_2', \texttt{Prog}_{else})$. Therefore, $\mathsf{SP}^c((C \wedge P_1') \vee (\neg C \wedge P_2'), \texttt{Prog}_{else})$ is abstracted by $\mathsf{CH}(R_3, R_4)$.

- We must now join the abstract mid-conditions at `L5` and `L8` to obtain the abstract mid-condition at `L9`. Since b is set to true along the "then" branch, and to false along the "else" branch, we can use b as the boolean condition in the abstract post-condition at `L9` to get $(b, \mathsf{CH}(R_1, R_2), \mathsf{CH}(R_3, R_4)))$. Note that this is more precise than simply taking the convex hull of the polyhedra $\mathsf{CH}(R_1, R_2)$ and $\mathsf{CH}(R_3, R_4)))$. The conditional nature of what each abstract element represents ("if C then P_1 else P_2") allows us to represent both P_1 and P_2 in the same abstract element, without taking their convex hull.

- Let $\widehat{P_1}$ and $\widehat{P_2}$ denote the convex polyhedra $\mathsf{CH}(R_1, R_2)$ and $\mathsf{CH}(R_3, R_4))$ respectively. On executing the statement at `L9`, i.e `z := z - 1`, starting from the pre-condition $(b, \widehat{P_1}, \widehat{P_2})$ computed above, the corresponding concrete post-condition at `L10` would be $\mathsf{SP}^c((b \wedge \widehat{P_1}) \vee$

$(\neg b \wedge \widehat{P_2}), \mathtt{z := z - 1})$. Since the assignment statement doesn't modify b, we can write the above as $(b \wedge \mathsf{SP}^c(\widehat{P_1}, \mathtt{z := z - 1}) \vee (\neg b \wedge \mathsf{SP}^c(\widehat{P_1}, \mathtt{z := z - 1})$. Therefore, if $\widetilde{P_1}$ and $\widetilde{P_2}$ represent the polyhedral abstractions of $\mathsf{SP}^c(\widehat{P_1}, \mathtt{z := z - 1})$ and $\mathsf{SP}^c(\widehat{P_1}, \mathtt{z := z - 1})$ respectively, the abstract post-condition at `L10` can be written as $(b, \widetilde{P_1}, \widetilde{P_2})$.

- To summarize, given $a = (C, P_1, P_2)$, the value of $F(a \sqcap \alpha(z > 0))$ giving the abstract post-condition for the loop body in our program, can be obtained as follows:

  1. Let R_1 be the polyhedral abstraction of $\mathsf{SP}^c((x = y) \wedge C \wedge P_1 \wedge (z > 0), \mathtt{Prog}_{then})$.
  2. Let R_2 be the polyhedral abstraction of $\mathsf{SP}^c((x = y) \wedge \neg C \wedge P_2 \wedge (z > 0), \mathtt{Prog}_{then})$.
  3. Let R_3 be the polyhedral abstraction of $\mathsf{SP}^c(C \wedge P_1 \wedge (z > 0), \mathtt{Prog}_{else})$.
  4. Let R_4 be the polyhedral abstraction of $\mathsf{SP}^c(\neg C \wedge P_2 \wedge (z > 0), \mathtt{Prog}_{else})$.
  5. Let $\widehat{P_1}$ and $\widehat{P_2}$ denote the convex polyhedra $\mathsf{CH}(R_1, R_2)$ and $\mathsf{CH}(R_3, R_4))$ respectively.
  6. Let $\widetilde{P_1}$ and $\widetilde{P_2}$ represent the polyhedral abstractions of $\mathsf{SP}^c(\widehat{P_1}, \mathtt{z := z - 1})$ and $\mathsf{SP}^c(\widehat{P_2}, \mathtt{z := z - 1})$ respectively.
  7. Then $F(a \sqcap \alpha(z > 0)) = (b, \widetilde{P_1}, \widetilde{P_2})$.

- Now to compute $G(a) = a_0 \cup F(a \sqcup \alpha(z > 0))$, we need to figure out how to compute the least upper bound (or an overapproximation of it) of a_0 with an abstract element of the form $(b, \widetilde{P_1}, \widetilde{P_2})$. Since we had a free choice of C in the representation (C, P, P) of a_0, it is beneficial to identify C with b, so that we only need to compute least upper bounds of the form $(b, P_1, P_2) \sqcup (b, P_3, P_4)$. As you can see, this is really trying to represent the set of concrete states $\{s : s \models (b \wedge P_1) \vee (\neg b \wedge P_2) \vee (b \wedge P_3) \vee (\neg b \wedge P_4)\}$. This set is simply $\{s : s \models (b \wedge (P_1 \vee P_3)) \vee (\neg b \wedge (P_2 \vee P_4))\}$. Therefore, it should come as no surprise that our choice of $(b, P_1, P_2) \sqcup (b, P_3, P_4)$ is $(b, \mathsf{CH}(P_1, P_3), \mathsf{CH}(P_2, P_4))$.

- So, now we have all the ingredients to compute the sequence of a_i's for our problem. Given below is a step-by-step calculation of the a_i's using the ideas described above. *This tabular form is really what you are expected to write in your solution.*

a_0	(b, P, P), where P is $(x > 0) \wedge (y > 0) \wedge (z > 0)$.
x_0	Same as a_0
x_1	Computing $G(a_0) = a_0 \sqcup F(a_0 \sqcap \alpha(z > 0))$:
	$R_1 = (z = 2x + 2) \wedge (x = y - 1) \wedge (x \geq 0) \wedge (y > 0) \wedge (z \geq 2)$
	$R_3 = (z = x + y) \wedge (x > 0) \wedge (y > 0) \wedge (z \geq 2)$
	$R_2 = R_1$ and $R_4 = R_3$
	$\widehat{P_1} = R_1$ and $\widehat{P_2} = R_3$
	$\widetilde{P_1} = (z = 2x + 1) \wedge (x = y - 1) \wedge (x \geq 0) \wedge (y > 0) \wedge (z \geq 1)$, and
	$\widetilde{P_2} = (z = x + y - 1) \wedge (x > 0) \wedge (y > 0) \wedge (z \geq 1)$
	Therefore, $F(a_0 \sqcap \alpha(z > 0)) = (b, \widetilde{P_1}, \widetilde{P_2})$
	Finally, $G(a_0) = a_0 \sqcup F(a_0 \sqcap \alpha(z > 0)) = (b, P, P) \sqcup (b, \widetilde{P_1}, \widetilde{P_2})$
	$= (b, P_3, P)$, where P_3 is $(x \geq 0) \wedge (y > 0) \wedge (z > 0)$
a_1	We'll use $\sqcup$ instead of ∇ for the first few steps.
	Hence, $a_1 = a_0 \sqcup x_1 = (b, P_3, P) = x_1$
x_2	Computing $G(a_1) = a_0 \sqcup F(a_1 \sqcap \alpha(z > 0))$:
	$R_1 = (z = 2x + 2) \wedge (x = y - 1) \wedge (x \geq 0) \wedge (y > 0) \wedge (z \geq 2)$
	$R_2 = R_1$
	$R_3 = (z = x + y) \wedge (x \geq 0) \wedge (y > 0) \wedge (z \geq 1)$
	$R_4 = (z = x + y) \wedge (x > 0) \wedge (y > 0) \wedge (z \geq 2)$
	$\widehat{P_1} = R_1$ and $\widehat{P_2} = R_3$
	$\widetilde{P_1} = (z = 2x + 1) \wedge (x = y - 1) \wedge (x \geq 0) \wedge (y > 0) \wedge (z \geq 1)$, and
	$\widetilde{P_2} = (z = x + y - 1) \wedge (x \geq 0) \wedge (y > 0) \wedge (z \geq 0)$
	Therefore, $F(a_1 \sqcap \alpha(z > 0)) = (b, \widetilde{P_1}, \widetilde{P_2})$
	Finally, $G(a_1) = a_0 \sqcup F(a_1 \sqcap \alpha(z > 0)) = (b, P, P) \sqcup (b, \widetilde{P_1}, \widetilde{P_2})$
	$= (b, P_3, P_4)$, where P_3 is $(x \geq 0) \wedge (y > 0) \wedge (z\ 0)$ and
	$P_4 = (x \geq 0) \wedge (y > 0) \wedge (z \geq 0)$
a_2	We'll use $\sqcup$ instead of ∇ for the first few steps.
	Hence, $a_2 = a_1 \sqcup x_2 = (b, P_3, P_4) = x_2$
x_3	Computing $G(a_2) = a_0 \sqcup F(a_2 \sqcap \alpha(z > 0))$:
	Note that $a_2 \sqcap \alpha(z > 0)$ represents the same concrete states as $a_1 \sqcap \alpha(z > 0)$
	Hence $G(a_2) = G(a_1) = x_2$
a_3	We'll use $\sqcup$ instead of ∇ for the first few steps.
	Hence, $a_3 = a_2 \sqcup x_3 = a_2 \sqcup x_2 = x_2 = a_2$

Since $a_3 = a_2$, the sequence of a_i's has stabilized and we have reached a post-fixpoint that represents a loop invariant. We can try to obtain a better abstract loop invariant by applying $G(\cdot)$ further. In this case, since $a_3 = a_2 = x_2$, therefore $G(a_3) = G(a_2) = x_2 = a_3$. Hence, a_3 is a fixpoint of G and represents an abstract loop invariant.

The abstract loop invariant is $(b, (x \geq 0) \wedge (y > 0) \wedge (z > 0), (x \geq 0) \wedge (y > 0) \wedge (z \geq 0))$.