Quiz

Synthesis, Analysis, and Verification 2012

Tuesday, April 17th, 2012

Last Name : _____

First Name : _____

Exercise	Points	Achieved Points
1	10	
2	10	
3	20	
4	30	
5	15	
6	15	
Total	100	

Problem 1: Relations (10 points)

Task a) Prove that for every $P \subseteq S$,

$$sp(P, r) = sp(S, \Delta_P \circ r)$$

Solution:

$$\begin{aligned} sp(S, \Delta_P \circ r) &= \{s' | \exists s.s \in S \wedge (s, s') \in (\Delta_P \circ r)\} \\ &= \{s' | \exists s, t.s = t \wedge t \in P \wedge (t, s') \in r\} \\ &= \{s' | \exists s.s \in P \wedge (s, s') \in r\} \\ &= sp(P, r) \end{aligned}$$

Task b) Prove that for every $P \subseteq S$,

$$wp(r, P) = \{x | sp(\{x\}, r) \subseteq P\}$$

Solution:

$$\begin{aligned} \{x | sp(\{x\}, r) \subseteq P\} &= \{x | \forall s'.s' \in sp(\{x\}, r) \rightarrow s' \in P\} \\ &= \{x | \forall s'.(\exists s.s \in \{x\} \wedge (s, s') \in r) \rightarrow s' \in P\} \\ &= \{x | \forall s'.(\exists s.s = x \wedge (s, s') \in r) \rightarrow s' \in P\} \\ &= \{x | \forall s'.(x, s') \in r \rightarrow s' \in P\} \\ &= wp(r, P) \end{aligned}$$

Problem 2: Fixpoints (10 points)

Let $r \subseteq A \times A$.

Task a) Let $D = \{r \mid r \subseteq A \times A\}$ and let $F_r : D \rightarrow D$ be defined as

$$F_r(s) = \Delta_A \cup s \circ r \circ s$$

Compute the least fixpoint of F according to the subset relation $\subseteq$ and give an alternative short description of it.

Solution: We want to compute the least fixpoint, say t. We claim that $r^* = t$. From

$$F_r(r^*) = \Delta_A \cup r^* \circ r \circ r^* = r^*$$

we see that it is a fixpoint. We can apply Tarski's fixpoint theorem, to get that $r*$ is the least fixpoint (with some work).

Task b) Now let

$$F_r(s) = s \circ r \circ s$$

What is the least fixpoint now?

Solution: Trying F_r on $\varnothing$: $F_r(\varnothing) = \varnothing \circ r \circ \varnothing = \varnothing$ we see that the empty set is a fixpoint, and since it is the least in the ordering, it's the least fixpoint.

Problem 3: Nice Invariants (20 points)

Suppose that $R(v, v')$ is a formula describing a piece of code and let $r = \{(v, v') \mid R(v, v')\}$ be the relation corresponding to R. Let $T(v, v')$ be a formula such that, for $t = \{(v, v') \mid T(v, v')\}$, the following holds:

$$(1) \quad \Delta \subseteq t$$
$$(2) \quad r \subseteq t$$
$$(3) \quad t \circ t \subseteq t$$

Let $P(v)$ and $Q(v)$ be formulas and assume that the following formula is valid:

$$(4) \quad P(v) \wedge T(v, v') \rightarrow Q(v')$$

We will say that $I(v)$ is a *nice invariant* if all of the following formulas are valid:

$$P(v) \rightarrow I(v)$$
$$I(v) \rightarrow Q(v)$$
$$I(v) \wedge R(v, v') \rightarrow I(v')$$

Task a) Prove or disprove that $sp(P, t)$ is a nice invariant.
Solution: We will prove it is a nice invariant.

- $P(v) \rightarrow sp(P, t)$
  We want to show
  $$P(v) \rightarrow \exists v'.P(v') \wedge (v', v) \in t$$
  Taking $v' = v$, the implication holds, since $(v, v) \in t$ by assumption.

- $sp(P, t)(v) \rightarrow Q(v)$
  We have
  $$\exists v_1.P(v_1) \wedge (v_1, v) \in t \rightarrow \exists v_1.Q(v) \rightarrow Q(v)$$
  as required from (4).

- $sp(P, t)(v) \wedge R(v, v') \rightarrow sp(P, t)(v')$
  We want to show
  $$[(\exists v_1.P(v_1) \wedge (v_1, v) \in t) \wedge (v, v') \in r] \rightarrow \exists v_2.P(v_2) \wedge (v_2, v') \in t$$
  Take $v_2 = v_1$ and by $(v_2, v') \in t \circ r \subseteq t \circ t \subseteq t$ the implication holds.

Task b) Prove or disprove that $wp(t, Q)$ is a nice invariant.
Solution:
We will prove it is a nice invariant.

- $P(v) \rightarrow wp(t, Q)(v)$
  We want to show
  $$\begin{aligned} & P(v) \rightarrow (\forall v_1.(v, v_1) \in t \rightarrow Q(v_1)) \\ \Leftrightarrow\ & \forall v_1.P(v) \rightarrow ((v, v_1) \in t \rightarrow Q(v_1)) \\ \Leftrightarrow\ & \forall v_1.P(v) \wedge (v, v_1) \in t \rightarrow Q(v_1) \end{aligned}$$
  which is valid by assumption.

- $wp(t, Q)(v) \to Q(v)$
  We want to show
  $$(\forall v'.((v, v') \in t \to Q(v'))) \to Q(v)$$
  Take $v' = v$ and since $(v, v) \in t$ the above holds.

- $wp(t, Q)(v) \wedge R(v, v') \to wp(t, Q)(v')$

$$\begin{aligned}
&(\forall v_1.(v, v_1) \in t \to Q(v_1)) \wedge R(v, v') \to (\forall v_2.(v', v_2) \in t \to Q(v_2)) \\
\Leftrightarrow &\forall v_2.(\forall v_1.(v, v_1) \in t \to Q(v_1)) \wedge R(v, v') \wedge (v', v_2) \in t \to Q(v_2)) \\
\Leftrightarrow &\forall v_2.(\forall v_1.(v, v_1) \in t \to Q(v_1)) \wedge (v, v_2) \in t \to Q(v_2))
\end{aligned}$$

  By taking $v_1 = v_2$ the condition holds.

Task c) How do the answers in a) and b) change if we replace the condition $t \circ t \subseteq t$ with the condition $r \circ t \subseteq t$? Recall that we define relation composition $\circ$ such that $r \circ t = \{(v, v') \mid \exists v''.(v, v'') \in r \wedge (v'', v') \in t\}$

Solution:

For part a) we have used $t \circ r \subseteq t \circ t \subseteq t$, which does not hold anymore. In fact, one can find a counterexample for the condition $sp(P, t)(v) \wedge R(v, v') \to sp(P, t)(v')$.

Problem 4: Hoare Triples and Loop Invariants (30 points)

Consider a programming language that supports integer variables, as well as variables that denote sets of integers and binary relations on integers (all integers are unbounded).
The command $\mathsf{lookup}(k, r)$ looks up a value v such that $(k, v) \in r$. If such value exists, it returns one such value as a singleton set $\{v\}$. If no such value exists, it returns the emptyset $\{\}$. (Note that, for each k, there can in general be zero, one, or more values v such that $(k, v) \in r$.)

Task a) Write a Hoare triple describing $\mathsf{lookup}(k, r1)$ in the form

$$\{\mathit{precondition}\} \;\; v1 = \mathsf{lookup}(k, r1) \;\; \{\mathit{postcondition}\}$$

where the precondition is as permissive (weak) as possible (so that it does not restrict the application of the lookup operation unnecessarily). Given as week precondition as you can find, specify the most precise postcondition that follows from the above description of how lookup should work.
Solution:

$$\{\top\} \quad v1 = \mathit{lookup}(k, r1) \quad \{(\exists v.(k, v) \in r1 \wedge v1 = \{v\}) \vee (\forall v.(k, v) \notin r1 \wedge v1 = \varnothing)\}$$

Task b) Consider the following program, where the variables r1,r are relations, v1,W are sets of integers, and k is an integer.

```
// Precondition: ∀i.∀v.(i,v) ∈ r → 0 ≤ i
r1 = r;
k = 0;
W = {};
while // invariant Inv
        (r1 != {})
{
  v1 = lookup(k,r1);
  if (v1 = {}) {
     k = k + 1
  } else {
     W = W ∪ v1;
     r1 = r1 \ ({k} × v1)
  }
}
// Postcondition: W = range(r)
```

We use the notation

$$\mathsf{range}(r) = \{v \mid \exists i.(i, v) \in r\}$$

Find an appropriate loop invariant, Inv, and use it to prove that, whenever we run the above program in a state that satisfies the Precondition, its final state satisfies the Postcondition. You need to explain why (1) the invariant holds initially in *all* states that satisfy the precondition, why (2) it is inductive (preserved on each execution of the loop body starting from *any* state satisfying only the invariant), and why (3) it can be used to prove the Postcondition. State each of these conditions as a Hoare triple, and prove it. Your proof of individual Hoare triples need not be very detailed.

Feel free to use any notation of sets, relations, and quantifiers in your invariants and Hoare triples. It is crucial that your invariant is correct (conditions (1),(2),(3) hold). Hint: using $r \setminus r_1$ as part of your loop invariant may be helpful.

Solution: There were two suggested invariants in the solutions of the students.

- $W = range(r \setminus r1)$
- $W \cup range(r1) = range(r)$

Notice that the first invariant is stronger than the second one. You can easily prove the following lemma: $(W = range(r \setminus r1) \wedge (r1 \subseteq r)) \rightarrow (W \cup range(r1) = range(r))$ However the following does not necessarily hold:
$(W \cup range(r1) = range(r) \wedge (r1 \subseteq r)) \rightarrow (W = range(r \setminus r1))$ Although the second invariant allows more freedom to the variables both of them are correct for the given program. Here we give the proof for the inductive case; entrance and exit can easily be proved. Notice that the second invariant can also be written as the following. The new form is a union of two sets in which the first is similar to the invariant:
$W = (range(r) \setminus range(r1)) \cup (W \cap range(r1))$
Let's show the three properties for the first invariant:

1. $$\{\forall i.\forall v.(i,v) \in r \rightarrow 0 \leq i\} \text{ r1 = r; k = 0; W = } \{W = range(r \setminus r1)\}$$
   We have $range(r \setminus r1) = range(\varnothing) = \varnothing = W$

2. $$\{W = range(r \setminus r1)\} \text{ loop body } \{W' = range(r \setminus r1')\}$$
   We have two cases to consider. In the first case, $v_1 = \{\}$ and $k' = k + 1$ and all other variables remain unchanged, hence the Hoare triple trivially holds.
   In the second case, $v_1 \neq \{\}$, $W' = W \cup v_1$ and $r_1' = r_1 \setminus (\{k\} \times v_1)$.
   $$\begin{aligned} range(r \setminus r_1') &= range(r \setminus (r_1 \setminus (\{k\} \times v_1))) \\ &= range((r \setminus r_1) \cup (r \cap (\{k\} \times v_1))) \\ &= range((r \setminus r_1) \cup (\{k\} \times v_1)) \qquad \text{since } (\{k\} \times v_1) \subseteq r \\ &= range(r \setminus r1) \cup range(\{k\} \times v_1) \\ &= W \cup v_1 \\ &= W' \end{aligned}$$

3. $$\{W = range(r \setminus r_1) \wedge r_1 = \varnothing\} \quad \{W = range(r)\}$$
   follows from $r \setminus r_1 = r$.

For the second invariant cases 1 and 3 are similarly easy, for the case 2, we want to prove $W' \cup range(r') = range(r)$ for $v_1 \neq \{\}$.

$$\begin{aligned} W' \cup range(r_1') &= W \cup v_1 \cup range(r_1 \setminus (\{k\} \times v_1)) \\ &= W \cup range(r_1) \cup v_1 \\ &= W \cup range(r_1) \\ &= range(r) \end{aligned}$$

Bonus Task c) Suppose that we modify the code above by inserting the assignment command 'k = k + 1' also in the second branch of 'if'. Does your original invariant still apply to the modified program?
Solution: Only if there are no duplicates.

Problem 5: Interval Analysis (15 points)

Consider interval analysis of a program with two integer variables x and y.
The state of the program is a map of the form $\{x \mapsto i, y \mapsto j\}$ where $i, j \in \mathbb{Z}$.
The abstract domain A associates an interval to each variable: it is the set of maps of the form

$$\{x \mapsto [l_x, u_x], y \mapsto [l_y, u_y]\}$$

with $l_x, l_y \in \mathbb{Z} \cup \{-\infty\}$ and $u_x, u_y \in \mathbb{Z} \cup \{\infty\}$.
The abstraction function α is such that given a set of concrete states S, $\alpha(S)(x)$ is the most precise interval containing all values of x found in S and $\alpha(S)(y)$ is the most precise interval containing all values of y found in S.
The concretization function γ is such that

$$\gamma(\{x \mapsto [l_x, u_x], y \mapsto [l_y, u_y]\}) = \{s.s(x) \in [l_x, u_x] \wedge s(y) \in [l_y, u_y]\}$$

Finally we define

$$sp^\sharp(a, c) = \alpha(sp(\gamma(a), c))$$

In the following questions the abstract postconditions need to be computed with respect to an arbitrary abstract precondition represented by $\{x \mapsto [l_x, u_x], y \mapsto [l_y, u_y]\}$. Please make sure that the abstract postconditions you give are as precise as possible.

Task a) Give the abstract strongest postcondition for each of the following statements

i) `y = 5*x^2 - 26*x +5`

ii) `x = x*y`

iii) `x = a*x + b*y`

Solution:

i) The polynomial $P = 5x^2 - 26x + 5$ is a vertical parabola with minimum at $x = 13/5$. Hence there are four cases:

- $l_x \leq u_x \leq 13/5$. In this case we have $sp^\sharp(c) = \{x \mapsto [l_x, u_x], y \mapsto [P(u_x), P(l_x)]\}$
- $l_x \leq 13/5 \leq u_x$ and $13/5 - l_x > u_x - 13/5$. In this case $sp^\sharp(c) = \{x \mapsto [l_x, u_x], y \mapsto [P(u_x), P(l_x)]\}$.
- $l_x \leq 13/5 \leq u_x$ and $13/5 - l_x \leq u_x - 13/5$. In this case $sp^\sharp(c) = \{x \mapsto [l_x, u_x], y \mapsto [P(l_x), P(u_x)]\}$.
- $13/5 \leq l_x \leq u_x$. In this case $sp^\sharp(c) = \{x \mapsto [l_x, u_x], y \mapsto [P(l_x), P(u_x)]\}$

ii) Let c be the statement `x = x*y`.

- If $l_x, l_y, u_x, u_y \leq 0$ then $sp^\sharp(c) = \{y \mapsto [l_y, u_y], x \mapsto [u_x * u_y, l_x * l_y]\}$
- If $l_x, l_y \leq 0$ and $u_x, u_y \geq 0$ then $sp^\sharp(c) = \{y \mapsto [l_y, u_y], x \mapsto [min(u_x * l_y, l_x * u_y), max(u_x * u_y, l_x * l_y)]\}$
- If $l_x, l_y, u_x, u_y \geq 0$ then $sp^\sharp(c) = \{y \mapsto [l_y, u_y], x \mapsto [l_x * l_y, u_x * u_y]\}$
- If $l_x, u_x \leq 0$ and $l_y, u_y \geq 0$ then $sp^\sharp(c) = \{y \mapsto [l_y, u_y], x \mapsto [l_x * u_y, u_x * l_y]\}$
- If $l_y, u_y \leq 0$ and $l_x, u_x \geq 0$ then $sp^\sharp(c) = \{y \mapsto [l_y, u_y], x \mapsto [l_y * u_x, u_y * l_x]\}$

iii) Let c be the statement `x = a*x + b*y`. Let $l_{ax}, u_{ax}, l_{by}, u_{by}$ be the bounds on $a * x$ and $b * y$ as determined above. Then $sp^\sharp(c) = \{x \mapsto [l_{ax} + l_{bx}, u_{ax} + u_{bx}], y \mapsto [l_y, u_y]\}$.

Task b) Use the rules you determined above to compute the abstract postcondition for the following program:

```
y = 5*x - 1;
x = x - 5;
y = y*x;
```

Solution:
Let x_0 and y_0 be the initial values of the variables. Observe that, at the end of the program, $x = x_0 - 5$ and $y = (5 * x_0 - 1)(x_0 - 5) = 5 * x_0^2 - 26 * x_0 + 5$. Hence we have that $sp^\sharp(p) = \{x \mapsto [l_x - 5, u_x - 5], y \mapsto [l'_y, u'_y]\}$ where l'_y and u'_y are as determined in question 1.1.

Problem 6: Predicate Abstraction (15 points)

Consider the set of predicates

$$\mathcal{P} = \{\text{false}, 0 \leq x, 0 \leq y, x \leq y\}$$

Let $A = 2^{\mathcal{P}}$. The meaning of a set of predicates $a \in A$, denoted $\gamma(a)$ is, as usual, the set of states that satisfies the conjunction of all predicates in a.
The precise semantics of a command `cmd` is the relation associated with `cmd`. For example, the precise semantics of the command

```
x = y; y = x + 1
```

in a program with two variables is the relation

$$\{((x, y), (x', y')) \mid x' = y \wedge y' = y + 1\}$$

For a given command `cmd` whose precise semantics is given by a relation r, let $sp^\#(a, \texttt{cmd})$ denote the least element $a' \in A$ such that $sp(\gamma(a), r) \subseteq \gamma(a')$.
As usual in programming languages, let `x++` denote a command that increments an integer variable `x` by 1 (assume that integer variables are unbounded).
Let $a_0 = \{0 \leq x, 0 \leq y, x \leq y\}$.
Compute the following values:

a) $\mathsf{sp}^\#(a_0, \texttt{x++})$

b) $\mathsf{sp}^\#(\mathsf{sp}^\#(a_0, \texttt{x++}), \texttt{y++})$

c) $\mathsf{sp}^\#(a_0, (\texttt{x++}; \texttt{y++}))$

Solution:

a) $\mathsf{sp}^\#(a_0, \texttt{x++}) = \{0 \leq x, 0 \leq y\}$

b) $\mathsf{sp}^\#(\mathsf{sp}^\#(a_0, \texttt{x++}), \texttt{y++}) = \{0 \leq x, 0 \leq y\}$

c) $\mathsf{sp}^\#(a_0, (\texttt{x++}; \texttt{y++})) = a_0$