CS615 Autumn 2007 End-semester Exam

Time: 180 mins | Total marks:

- ***You must write your answers only in the spaces provided.***
- ***The exam is open book and notes.***
- ***Results/proofs covered in class/problem sessions/assignments may simply be cited, unless specifically asked for.***
- ***Unnecessarily lengthy solutions will be penalized.***
- ***If you need to make any assumptions, state them clearly.***
- ***Do not copy solutions from others or indulge in unfair means.***

1. Consider the following program P in the language studied in class.

```
L1: x  := y + z;
L2: d2 := 0;
L3: x1 := z;
L4: d1 := z;
L5:

L6: while (x < N) {
L7:   x1 := x;
L8:   d2 := d1;

L9:   y := y + z;
L10:  z := y + z;
L11:  x := y + z;

L12:  d1 := x - x1;
L13:}
L14:
```

We wish to prove the validity of the Hoare triple $\{\mathsf{True}\}\ P\ \{\mathtt{d1} = \mathtt{d2} + \mathtt{x1}\}$.

Using only conjunctions of linear equalities and inequalities between program variables (no quantified formulae, no auxiliary variables), write an invariant for each location `L2` through `L14` in the above program. You must write your invariant only in the spaces provided for this purpose below.

Recall that an invariant at location L_i is a formula on program variables that holds whenever the program reaches location L_i. Your invariants should be such that:

- The invariant at `L14` implies the postcondition ($\mathtt{d1} = \mathtt{d2} + \mathtt{x1}$).
- For all locations other than `L1, L6, L7, L14`, the invariant at L_i must be derivable using only the invariant at L_{i-1}, and the "assignment" and "implied" rules of Hoare Logic.
- The invariant at `L6` must be implied both the invariants at `L5` and `L13`.
- The invariant at `L7` must be implied by the conjunction of the invariant at `L6` and ($\mathtt{x} < \mathtt{N}$).

You need not write the strongest invariant at each location. Your invariants should however be strong enough to permit a Hoare Logic proof of $\{\text{True}\}\ P\ \{\texttt{d1} = \texttt{d2} + \texttt{x1}\}$.

For your convenience, all rules of Hoare Logic that are relevant for proving validity of the above Hoare triple are given below. Note, however, you are not being asked to give the complete Hoare Logic proof.

$\dfrac{}{\{\phi[E/x]\}\ \texttt{x:=E;}\ \{\phi\}}$	Assignment
$\dfrac{\{\phi\}\ P_1\ \{\phi_1\} \quad \{\phi_1\}\ P_2\ \{\psi\}}{\{\phi\}\ P_1;P_2\ \{\psi\}}$	Composition
$\dfrac{\{\phi \wedge B\}\ P\ \{\phi\}}{\{\phi\}\ \texttt{while(B) P;}\ \{\phi \wedge \neg B\}}$	Partial while
$\dfrac{\phi_1 \rightarrow \phi \quad \{\phi\}\ P\ \{\psi\} \quad \psi \rightarrow \psi_1}{\{\phi_1\}\ P\{\psi_1\}}$	Implied

Hint: Before jumping to provide a solution, consider running the program for a few iterations through the loop to understand the pattern of dependency of variables.

(a) *[13 × 2 marks]* Write your invariants here:

L1: True

L2:

L3:

L4:

L5:

L6:

L7:

L8:

L9:

L10:

L11:

L12:

L13:

L14:

(b) *[9 marks]* Show how the invariant at `L6` is implied both the invariants at `L5` and `L13`.

(c) Consider the following program:

```
L1:  x := 4 + z;

L2:  while (x > z) {
L3:    x := y + z;
L4:    if (?) { y := y - 2;
L5:    }
L6:    else { y := y - 1;
L7:    }
L8:    z := y + z;
L9:  }
L10:
```

We wish to analyze this program using the technique of abstract interpretation. We will use the abstract domain of convex polyhedra for our analysis.

In the following subquestions, *you must represent each convex polyhedron by a system of* 2 *or fewer linear inequalities/equalities on the program variables* `x`*,* `y` *and* `z`*. All polyhedra arising in this question can indeed be accurately represented in this manner.* The constraints represented by a set of linear inequalities are assumed to be conjoined ("and"-ed) in order to obtain the desired polyhedron. For example, the polyhedron representing the state at `L2` when we hit `L2` for the first time is given by the single linear equality $(x = 4 + z)$.

i. *[7 × 2 marks]* Give the convex polyhedron at each location from `L3` through `L9` after analyzing one iteration of the while loop. Your convex polyhedra must be represented as described above.

`L1:` True

`L2:` $(x = 4 + z)$

`L3:`

`L4:`

`L5:`

`L6:`

`L7:`

`L8:`

`L9:`

ii. Suppose we decide to use the *lub* operator (convex hull for convex polyhedra) at the loophead (`L2`) to compute the loop invariant. Recall that such an approach has the risk that the abstract analysis may not terminate in general.

A. *[3 + 3 marks]* What are the convex polyhedra at locations `L2` and `L9` after analyzing two iterations of the loop? Briefly justify your answer.

At `L2`:

Justification:

At `L9`:

Justification:

iii. *[5 marks]* Will be abstract analysis of the above program using *lub* to compute the loop invariant at `L2` terminate? Briefly justify your answer in not more than three sentences.

2. Consider the following program that manipulates the heap.

```
// declarations etc.

typedef struct cell_t {
 int val;
 struct node *left, *right;
} cell;

cell *curr;
```

```
// actual code is here

L1: while (curr != NULL) {
L2:   if (curr->val == 0) {
L3:      curr := curr->lchild;
L4:   }
L5:   else { curr->val := 0;
L6:          curr := curr->rchild;
L7:   }
L8: }
```

We wish to study the effect of this program on the ***concrete*** heap structure shown in Fig. 1, using the abstraction of shape graphs obtained by three valued logic analysis studied in class. We will use the following predicates for constructing shape graphs. In the following table, v, v_1, v_2 denote heap cells or a special cell denoting the NULL value.

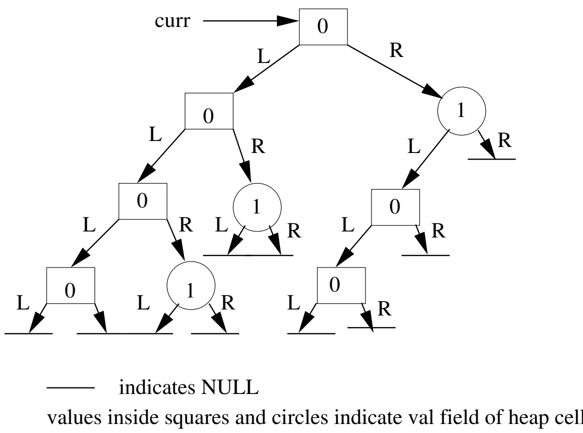

Figure 1: Concrete Heap Structure

Predicate	**Unary/Binary**	**True if and only if**
$val(v)$	Unary	$v.val == 0$
$curr(v)$	Unary	$curr$ points to cell v
$Null(v)$	Unary	v is a special unique cell denoting NULL value
$LNull(v)$	Unary	v's left child is NULL
$RNull(v)$	Unary	v's right child is NULL
$left(v_1, v_2)$	Binary	v_2 is v_1's left child
$right(v_1, v_2)$	Binary	v_2 is v_1's right child

In addition, we have the unary sm (summary) predicate, which evaluates to False for an abstract heap cell if the cell represents a unique concrete heap cell. Otherwise, if the abstract cell represents potentially multiple concrete heap cells, all of which agree on the unary predicates in the above table, sm evaluates to "?" or $\frac{1}{2}$ for the abstract cell.

(a) *[10 marks]* Give the abstract shape graph corresponding to the concrete graph shown in Fig. 1. Use the following notation when constructing the abstract shape graph:

- Cells for which val evaluates to True must be indicated as squares. Cells for which val evaluates to False must be indicated as circles.

- Cells for which $LNulL$ evaluates to True must be labeled **L0**, those for which $LNulL$ evaluates to False must be labeled **L1**.
- Cells for which $RNulL$ evaluates to True must be labeled **R0**, those for which $LNulL$ evaluates to False must be labeled **R1**. Thus a cell can have multiple labels, e.g. **L0,R1**.
- There must be only one cell for which $Null$ evaluates to True. You must show this as a shaded cell shaped like a diamond.
- Arrows denoting right links must be labeled **R** and arrows denoting left links must be labeled **L**.
- Summary nodes must be indicated by double-circling/double squaring them (as done in class).
- Dotted arrows must be used to denote valuations of $left$ or $right$ predicates that evaluate to "?" or $\frac{1}{2}$. Thus, if $left(u, v) =$ "?", then you must have a dotted arrow labeled **L** from u to v.
- $curr$ must always point to a unique node. Thus, the $curr$ arrow must never be dotted.
- No two abstract cells must have the same valuation of all unary predicates.

Abstract shape graph for Fig. 1:

(b) *[5 marks]* Give another concrete heap structure (different from Fig. 1) that would give rise to the same abstract shape graph as that obtained from Fig. 1.

(c) Using the same notation for abstract shape graphs as described above, give the abstract shape graph (or set of abstract shape graphs) at `L1` after the first two iterations through the loop. Note that you must no longer refer to the concrete heap structure in Fig. 1. Your entire analysis must be based on the abstract shape graph you obtained above. Thus, the analysis you are doing must apply not only to the concrete heap structure in Fig. 1, but also to all other concrete heap structures that give rise to the same abstract shape graph as obtained above.

i. *[5 marks]* Abstract shape graph after first iteration through loop when analyzing abstract shape graph for Fig. 1:

ii. *[5 marks]* Abstract shape graph after second iteration through loop when analyzing abstract shape graph for Fig. 1a:

iii. *[10 marks]* If we continued the above analysis until all abstract shape graphs that can arise at `L1` have been obtained, how many abstract shape graphs will be there in the loop invariant at `L1`. Briefly justify your answer.

3. *[10 marks]* Consider the program of Question 1 again. Suppose a student has correctly identified the set of invariants at each program location, as asked in Question 1. Let the invariant identified by the student for location L_i be I_i. Each invariant I_i is a conjunction of linear equalities/inequalities, as required by Question 1. Each such linear equality/inequality can also be considered a predicate potentially usable in a Boolean program analysis based approach.

   Thus, a potential approach to doing Boolean program analysis of the program in Question 1, is to first identify the invariants as asked in Question 1, and to then use the set of predicates in invariant I_i as the set of predicates to be tracked at location I_i. A Boolean program can then be constructed according to the predicates tracked at each location.

   If we follow this idea and construct a Boolean program corresponding to the program of Question 1, do you think the Boolean program based analysis will succeed in proving the Hoare triple $\{\mathtt{True}\}\ P\ \{\mathtt{d1} = \mathtt{d2} + \mathtt{x1}\}$? Give brief justification (no more than 10 sentences) for your answer. Note that a Boolean program based analysis will assert the post condition at location `L14` and then check the reachability of the "error" location.