Exam Program Verification 2010/2011

9 nov. 2010, 9:00–12:00, BBL-165

Lecturer: Wishnu Prasetya

1. **Hoare logic** [2 pt].

   Background

   Let's assume we add the type Set to uPL. We introduce the type below for representing graphs.

```
type Node  = int       ;  // We'll just represent a node with its ID

type Graph = record{n:int, sucs : (Set Node)[]} ;
```

   A value `g:Graph` represents a directed graph with $N =$ `g.n` number of nodes. We'll identify the nodes with numbers $0..N-1$.

   For each node `i`, `g.sucs[i]` gives the set of all 'successor nodes' of `i`; these are nodes that you can reach by following a single arrow in the graph.

   You get the following utility function/procedure:

   - `nodes(g) : Set int` returns the set of all nodes belonging to `g`.
   - `pull1(S)` returns an arbitrary element from `S`, provided it is non-empty; the element is removed from `S`.

   The program below will explore all nodes in `g` which are reachable from a given node `r`. These reachable nodes will be returned in a set.

```
explore(g:Graph, r:int) : Set int {

  V,S : Set int ;

  V = ∅ ;  // V maintains all nodes we have visited
  S = {r} ;

  while S ≠ [] do {
     int i ;
     i := pull1(S) ;
     V := V ∪ {i} ;
     S := S ∪ (g.sucs[i] / V) ;
  } ;

  return V
}
```

   Tasks

   (a) Give a formal specification for the program above.

   Answer:

$$\{\ \mathtt{r} \in \mathtt{nodes(g)}\ \}\quad \mathtt{explore(g,r)}\quad \{\ \mathtt{return} = \mathtt{r}^*\ \}$$

   where r^* denotes the set of all nodes that are reachable from r.

(b) Provide a realistic loop invariant and termination metric.

Answer:

$$Inv: \quad (V \cup S^*) = r^* \ \wedge \ V \subseteq nodes(g) \ \wedge \ (V \cap S = \emptyset)$$

where S^* denote the set of all nodes that are reachable from any node in S.

With the first conjuct we can prove EC trivially. The other conjuncts are needed to prove termination.

Termination metric: `g.n` $- \#$`V`.

(c) Provide an argument that $I \wedge \neg g$ implies your post-condition, where I is your invariant and g is the negation of the loop's guard above.

Answer:

well, with the above invariant, this is trivial.

(d) Provide an argument that your choice of termination metric does indeed implies that the loop terminates.

Answer:

The 2nd conjunct of Inv implies that `g.n` $- \#$`V` $\geq$ `0`.

Every iteration adds an element to `V`. The 3rd conjunct of Inv implies that moving an element from S to V will really add a new element to V, therefore cause the termination metric to decrease.

2. **LTL** [3 pt].

Consider a Kripke structure M with finite number of states, with labels from this set $Prop = \{p, q, r\}$. For simplicity we assume all M's executions are infinite.

(a) We want M's abstract executions to always start with a prefix satisfying a certain regular expression. Consider these expressions:

i. p^*q^*r
ii. p^*qq^*r
iii. p^*pq^*r

Give LTL formulas expressing each of the above requirements.

(b) Give a Buchi automaton that accepts the same language (over the above $Prop$) as $(\Diamond p)$ **U** q.

Hint: look first that the kinds of sentences satisfying the LTL.

(c) Suppose we have converted M to a Buchi automaton B, and suppose we have another Buchi automaton C representing $\neg\phi$ for some LTL formula ϕ.

Give a definition for $B \cap C$ which we would need for LTL model checking.

3. **Model checking** [2 pt]

Consider a potentially large but *finite* state automaton M with . For simplicity we assume all M's executions are infinite.

(a) Suppose we mark some states of M as error states: it is an error if there is an execution of M that passes such a state.

Give an algorithm to verify that M avoids a set E of error states. You can use the algorithm `explore` from No. 1 above.

Answer:

'Check if $E \cap explore(M, s_0) \neq \emptyset$.

(b) Suppose we mark some states of M as 'progress states'. This is a requirement that every infinite execution of M has to pass at least one such state infinitely many times. Give an algorithm to verify that M satisfies a given set P of progress states.

Answer:

We can for example use the program `explore(M, s0)` again, but with a modification. Whenever we pull i from S, run a DFS to check for cycle. Each time a cycle is detected we check if the cycle contains a state from P. If not, we have a violation.

4. **CTL** [2 pt]

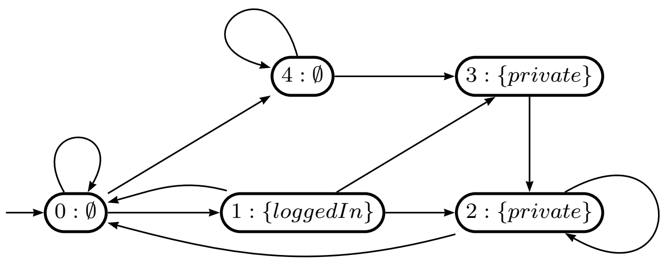
Consider a Kripke structure over $Pred = \{loggedIn, private\}$. For simplicity we assume M only has one initial state s_0 and all M's executions are infinite.

(a) Express the following requrements in CTL:

i. From any state in M's computation tree, there is a path leading to a state marked with *private*.
Answer: $AG(EF\ private)$

ii. M's computation tree contains no path from the root to a state decorated with *private*, that didn't pass a state decorated with *loggedIn*.
Answer: $\neg E((\neg pass \wedge \neg private)\ \mathbf{U}\ private)$

(b) Now consider the following M. The states are numbered 0..4; state 0 is the initial state. The labelling of every state is given after the ':'.



Show how the states is labelled after you run the model checking of the property $E(loggedIn\ \mathbf{U}\ private)$.

(c) The symbolic version of the model checking algorithm uses ordered BDDs to calculate the labeling. Give an ordered BDD describing your set of states labelled by $E(loggedIn\ \mathbf{U}\ private)$.

(d) What does the term 'ordered' here mean? Why do you want the BDDs to be ordered?

5. **HOL** [1 pt]

In HOL a tactic is a function of type (in Haskell notation):

$$\texttt{goal} \ \rightarrow \ ([\texttt{goal}], ([\texttt{thm}] \rightarrow \texttt{thm}))$$

where $\texttt{goal} = ([\texttt{term}], \texttt{term})$. You will give an example of how we can implement a tactic by reversing a rule. To make it simple, consider this version of the $\texttt{MP} : \texttt{thm} \rightarrow \texttt{thm} \rightarrow \texttt{thm}$ (Modus Ponens) rule:

$$\texttt{MP}\ (A \vdash t \Rightarrow u)\ \ \frac{A \vdash t}{A\ \vdash\ u}$$

Now, give a definition of the tactic that corresponds to `MP` above. So, it should do this:

$$\texttt{MP\_TAC}\ (A \vdash t \Rightarrow u)\ \ \frac{A\ ?\!-\ u}{A\ ?\!-\ t}$$

Above, `MP_TAC` has the type `thm` → `tactic`. You can write it in Haskell. You can assume to have sufficient functions to destruct your terms.