Exam-2 Program Verification 2019/2020

RUPPERT-C, 5th Nov. 2019, 17:00 - 20:00

Lecturer: Wishnu Prasetya

1. ... [1.5 pt].

2. **Theory** [1 pt].

   Let $M = (S, \{s_0\}, R, Prop, V)$ be a Kripke structure where S is a finite set of states, $s_0 {\in} S$ is M's initial state, $R : S {\rightarrow} \mathbf{Pow}(S)$ describes the transitions between the states, $Prop$ is a set of state predicates, and $V : S {\rightarrow} \mathbf{Pow}(Prop)$ describes the labelling of the states with members from $Prop$.

   Consider the following variation of LTL called LTL$^{\text{bounded}}$. Let ϕ and ψ represent LTL$^{\text{bounded}}$ formulas.

$$
\begin{array}{lcll}
\phi & ::= & p & \text{, where } p \in Prop \text{ (so, } p \text{ is a state predicate)} \\
 & | & \neg\phi & \\
 & | & \phi \wedge \psi & \\
 & | & \mathbf{X}\phi & \\
 & | & \phi\ \mathbf{U}^{*}\ \psi & \\
 & | & \phi\ \mathbf{U}^{\leq k}\ \psi & \text{, where } k \text{ is a concrete non-negative integer}
\end{array}
$$

   - The meaning of $\phi\ \mathbf{U}^{\leq k}\ \psi$ is the same as $\phi\ \mathbf{U}\ \psi$ in the standard LTL, except that the future ψ should happen within at most k steps.
   - The meaning of $\phi\ \mathbf{U}^{*}\ \psi$ is the same as $\phi\ \mathbf{U}^{\leq k}\ \psi$ with $k = \infty$.
   - The meaning of other constructs is the same as in the standard LTL.

   Your task: give a formal definition of what $\sigma, i \models \phi$ means for all the above constructs of LTL$^{\text{bounded}}$.

   Answer: Check the Lecture Notes for the formal definition if the standard LTL operators. You should then be able to figure out how to define the new operators.

3. **Buchi** [1.5 pt].

   Give for each LTL formula below, a Buchi automaton that equivalently describes the formula. Do not forget to specify what are the initial and accepting states are. Also, indicate whether you use a standard or generalized Buchi.

   Assume $Prop = \{p, q\}$.

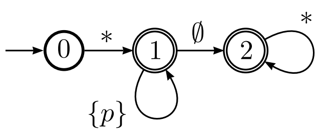
   (a) $\mathbf{X}(p \mathbf{\ U\ } q)$

   (b) $\neg\mathbf{X}(p \mathbf{\ U\ } q)$

   Answer: It is easier to first rewrite the property by working out the negation: $\neg\mathbf{X}(p \mathbf{\ U\ } q)$ can be rewritten to $\mathbf{X}\neg(p \mathbf{\ U\ } q)$, which can again be rewritten to:

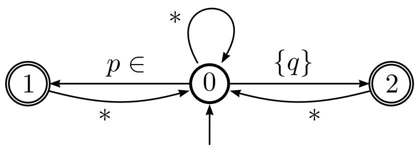
   $$\mathbf{X}((p \wedge \neg q) \mathbf{\ W\ } (\neg p \wedge \neg q))$$

   The Buchi automaton representing it is below. The initial state is 0. It has two accepting states: $\{1, 2\}$.



   (c) $(\Box\Diamond p) \wedge (\Box\Diamond(q \wedge \neg p))$

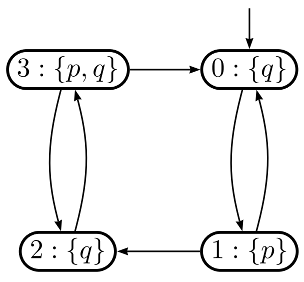
   Answer: Below we give a generalized Buchi automaton that describes the formula above. The initial state is 0. We have two groups of accepting states: $\{1\}$ and $\{2\}$.



   So, why is one of the labels is denoted as a set $\{q\}$ and the other as $p \in$?

4. **LTL model checking** [1.5 pt].

   Consider the following Kripke structure K that you can think as modelling some program. $Prop = \{p, q\}$. The initial state is 0.



   Describe the steps of LTL model checking to verify whether the following LTL property ϕ holds on K:

   $$(p \vee q) \mathsf{\ U\ } (p \wedge q)$$

   Do **provide your resulting intersection automaton**.

   Answer:

(a) Construct the Buchi automaton representing the negation of the formula to verify. Recall first that $\neg(f\ \mathbf{U}\ g) = (f \wedge \neg g)\ \mathsf{W}\ (\neg f \wedge \neg g)$. So, $\neg\phi$ is equivalent to:

$$((p \wedge \neg q) \vee (q \wedge \neg p)\ \mathsf{W}\ (\neg p \wedge \neg q)$$

which we can represent with the following Buchi B:

$$\begin{tikzcd}
{} \arrow[r] & a \arrow[loop above, "\{q\}"] \arrow[loop below, "\{p\}"] \arrow[rr, "\emptyset"'] & & b \arrow[loop right, "*"]
\end{tikzcd}$$

(b) We also convert the program K to a Buchi automaton. It should accept the same language as K. I covert it to the Buchi shown below. All states are accepting.

$$\begin{tikzcd}
 & {} \arrow[d] \\
3 \arrow[r, "\{p,q\}"'] \arrow[d, bend right, "\{p,q\}"'] & 0 \arrow[d, bend right, "\{q\}"'] \\
2 \arrow[u, bend right, "\{q\}"'] & 1 \arrow[l, "\{p\}"'] \arrow[u, bend right, "\{p\}"']
\end{tikzcd}$$

(c) Calculate the intersection between the two Buchi automata. This gives the following automaton. All its states are accepting (can you figure out why?):

$$\begin{tikzcd}
 & {} \arrow[d] \\
(3,a) & (0,a) \arrow[d, bend right, "\{q\}"'] \\
(2,a) \arrow[u, "\{q\}"'] & (1,a) \arrow[l, "\{p\}"'] \arrow[u, bend right, "\{p\}"']
\end{tikzcd}$$

(d) Conduct the verification itself. ϕ is valid on K if the language of the above intersection automaton is empty. However, this is not the case. The automaton has at least one accepting execution; I'll leave it to you to figure out which one.

5. **SPIN/Promela** [1 pt].

Consider a Promela system consisting of **three** processes: `P(0)`, `P(1)`, and `P(2)`. They are defined below. Let's call the system of these three processes *Sys*.

```
#define NEXT(i) i+1
#define FREE 127
byte a[4] ;
byte lock[4] ;

proctype P(byte i) {
  byte tmp ;
  do
  :: { atomic { (lock[i]==FREE  && lock[NEXT(i)]==FREE) ;
                lock[i] = i ;
                lock[NEXT(i)] = i } ;
       if
```

```
        :: (a[i]>a[NEXT(i)]) ;
            /* swap a[i] and a[i+1]: */
            tmp = a[i] ; a[i] = a[NEXT(i)] ; a[NEXT(i)] = tmp
        :: else -> skip
        fi ;
        lock[i] = FREE ;
        lock[NETXT(i)] = FREE
      }
   od
 }
```

We claim the system *Sys* as defined above will concurrently sort the array **a** ascendingly.

Questions:

(a) Give an LTL specification that would fully capture the correctness of *Sys*.

(b) Consider the progress property $\Diamond(\mathtt{a[0]} \leq \mathtt{a[1]})$. You might expect this property to be valid on *Sys*, but unfortunately it is not, unless you insist on some fairness assumption. Please explain: what kind of fairness assumption would we need here?

Answer: We can require strong fairness for every atomic action in the system. But since every process here is sequential (every process in Promela is sequential), it is enough to require weak fairness **among the processes** (to be distinguished with WF for every action). That is, of a process $P(i)$ persistently has one or more enabled actions to execute, then eventually $P(i)$ will get a turn to execute one of its enabled actions.

(c) The model above is unrealistic because it allows each process `P(i)` to acquire multiple locks in a single atomic statement. Can you remedy this? You can assume that Promela statements of the form $(e_x); x{=}d_x$ can be implemented atomically, provided e_x and d_x are expressions that only mention x as variables.

A sequential solition is **not** desired.

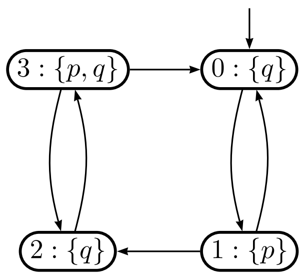
Answer: We can split the **atomic**{..} block in the original model to two parts:

atomic{ $(lock[i] == FREE)$; $lock[i] = i$ } ;
atomic{ $(lock[NEXT(i)] == FREE)$; $lock[NEXT(i)] = i$ }

This split can theory cause some processes to deadlock because they hold each other locks. However in this particular system this fortunately cannot be the case as the processes cannot lock each other in a cycle (we will not prove this, as the proof is not requested, but you can verify this in Spin yourself).

6. **CTL model checking** [1.5 pt].

Consider again the Kripke structure K from question No 4. So:



Describe the steps of CTL model checking in order to verify if the CTL property:

$$\phi \;=\; \mathbf{E}((p \vee q)\;\mathbf{U}\;((p \wedge q) \wedge \mathbf{AX}q))$$

holds on K.

Do show in your explanation how you calculate $W_{\mathbf{E}((p \lor q)\ \mathbf{U}\ ((p \land q) \land \mathbf{AX}q))}$, and what W_ϕ finally is.

Answer:

We first label the states of K with the sub-formulas of ϕ. A state s is labelled with ψ if and only if ψ holds on the computation tree rooted at s. We work this labelling recursively over the structure of ϕ:

(a) $W_{p \lor q} = \{0, 1, 2, 3\}$

(b) $W_{p \land q} = \{3\}$

(c) $W_{\mathbf{AX}q} = \{1, 2, 3\}$

(d) So, $W_{p \land q \land \mathbf{AX}q} = W_{p \land q} \cap W_{\mathbf{AX}q} = \{3\}$

(e) Finally, to calculate $W_\phi = W_{\mathbf{E}((p \lor q)\ \mathbf{U}\ ((p \land q) \land \mathbf{AX}q))}$ we iterate until we get a fix point:

$Z_0 = W_{p \land q \land \mathbf{AX}q} = \{3\}$
$Z_1 = Z_0 \cup \{s \mid s{\in}W_{p \lor q} \land (\exists t :: t{\in}Z_0 \land t \text{ is a successor of } s)\} = \{3\} \cup \{2\}$
$Z_2 = Z_1 \cup \{s \mid s{\in}W_{p \lor q} \land (\exists t :: t{\in}Z_1 \land t \text{ is a successor of } s)\} = \{2, 3\} \cup \{1, 2, 3\}$
$Z_3 = Z_2 \cup \{s \mid s{\in}W_{p \lor q} \land (\exists t :: t{\in}Z_2 \land t \text{ is a successor of } s)\} = \{1, 2, 3\} \cup \{0, 1, 2, 3\}$
Z_3 contains all the states in K, so it must be a fix point.

So, $W_\phi = \{0, 1, 2, 3\}$.

Next, we check whether W_ϕ contains K's initial state. It is, and therefore we conclude that ϕ must be valid on K.

(THERE ARE MORE QUESTIONS; next page)

7. **Symbolic model checking** [1.5 pt].

Let the Boolean formula:

$$R(x,y,x',y') \;=\; \bar{x}yx' \vee x\bar{y}\bar{x'}$$

encode the transitions of a program K. In this formula, the values of x, y represent the source states of a given transition, and x', y' represent the transition's destination states.

xy means the conjunction $x \wedge y$, and $\bar{x}$ means the negation $\neg x$.

(a) Suppose the formula xy describes all states of K where q holds. Give the Boolean formula that represents **EX** q.

Answer: The formula the describes **EX** q is:

$(\exists x', y' :: \; R(x,y,x',y') \wedge x'y')$, which is:

$(\exists x', y' :: \; (\bar{x}yx' \vee x\bar{y}\bar{x'}) \wedge x'y')$, which can be simplified to:

$(\exists x', y' :: \; \bar{x}yx'y')$, which can be simplified to: $\bar{x}y$

(b) As above, but now give the Boolean formula that represents **AX** q.

Answer: The formula the describes **AX** q is:

$(\forall x', y' :: \; R(x,y,x',y') \Rightarrow x'y')$, which is:

$(\forall x', y' :: \; \underbrace{(\bar{x}yx' \vee x\bar{y}\bar{x'})}_{left} \Rightarrow \underbrace{x'y'}_{right})$

This is a bit hard to simplify for human. But we can notice that the formula $left$ does not constrain y', and therefore cannot imply y' on the *right*. For example, consider these values for x, y, x':

$$x{=}0, \;\; y{=}1, \;\; x'{=}1$$

This values would make $left$ true. However with $y'{=}0$ we can't make *right* true. More generally, for values of x, y that make $left$ true, it is not possible to make the implication $left \Rightarrow right$ true for all x', y'.

However, for values of x, y that would make $left$ false, $left \Rightarrow right$ formula would true regardless the values of x', y'. The combinations that would make $left$ false is equivalently described by $xy \vee \bar{x}\bar{y}$.

In other words, we can simplify the above $\forall$ formula to just $xy \vee \bar{x}\bar{y}$.

(c) Construct the reduced OBDD (Ordered Binary Decision Diagram) representing R, using x', y', x, y as the ordering.

Answer:

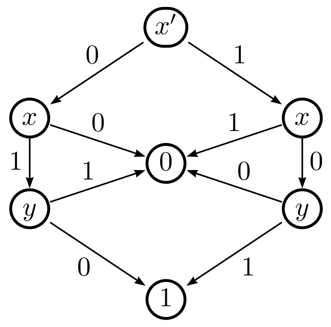
It happens that the formula R is in the DNF form, and furthermore I actually choose an ordering that favors the clauses in this DNF. Note that in general your formula is not necessarily in the DNF form, and trying to construct its BDD by first trying to construct a DNF may be problematical as the size of the DNF could be pretty large.

Anyway, R is in DNF in this example. Let me also rewrite it a bit to make it easier for you to align the clauses with the given ordering. So, R can be written as as follows, by re-ordering the variables appearing in its clauses:

$$R(x,y,x',y') \;=\; x'\bar{x}y \vee \bar{x'}x\bar{y}$$

The reduced OBDD representing this formula is given below. "Reduced" means that no two sub-graphs of this BDD are isomorphic, and no node has the same low- and high-child (because then the node would represent the same formula as its child, and hence redundant).

Notice that the BDD has exactly two paths that lead to the leaf "1". These two paths correspond to the two clauses we see above.



8. **Challenge** [0.5 pt].

   ....