Exam-2 Program Verification 2017/2018

BBG-001, 8th Nov. 2017, 9:00 - 12:00

Lecturer: Wishnu Prasetya

1. **Formalizing Properties** [1.5 pt].

   Let $P, Q, ... \in Process$ and $r, s, ... \in Resource$ be processes in a system and resources that the processes may want to use. Formalize the following properties; you can use either LTL, CTL, or CTL*. You can introduce state predicates to abstractly capture needed concepts, e.g. $req(P, r)$ can be a state predicate modeling the fact that P is currently requesting the resource r, if the predicate is true, and otherwise it is not currently requesting r.

   (a) P and Q are not allowed to access r at the same time.
   Answer: $\neg\Diamond(access(P, r) \wedge access(Q, r))$

   (b) The system should be weakly fair. That is, if a process P repeatedly requesting a resource (it does not need to persistently maintain the request), P should eventually get access to the resource.
   Answer: This should hold for all $P \in Process$ and $r \in Resource$: $\Box((\Box\Diamond req(P, r)) \rightarrow \Diamond access(P, r))$

   (c) Whenever P is requesting access to r, if after maintaining this request for some time it still does *not get the access*, it should be possible for P to cancel this request.
   Answer: Notice first that the part "it should be possible to cancel" is an 'existential' sort of requirement. It does not require that P should always cancel its request; it only requires that cancelling is possible. In other words, there should exist a possibility to cancel. Such an existential property can be expressed in CTL (thus also in CTL*), but not in LTL (since in LTL all properties are required to hold on *all* executions).
   So, in CTL or CTL*:
   $\mathbf{AG}(req(P, r) \rightarrow \mathbf{E}[req(P, r) \wedge \neg access(P, r)\ \mathbf{U}\ \neg req(P, r)])$

   (d) At all times, there should be at least one resource that is not used by any process.
   Answer: $\Box((\exists r \in Resource.\ (\forall P \in Process.\ \neg access(P, r))))$

   (e) It should be possible for P to use s right after it has used r.
   Answer: Let abbreviate $access(P, r)$ with just r; and similarly for s. In CTL*:
   $\mathbf{E}(\mathbf{F}(r \wedge (r\ \mathbf{U}(\neg r \wedge s))))$

2. **Expressing LTL as an FSA** [1.5 pt].

   Give for each of the LTL formula below, a Buchi automaton that equivalently describes the formula. You can use either ordinary or generalized Buchi automata. For each automaton, please explicitly **specify its groups of accepting states**.

   (a) $\neg(p\ \mathbf{U}\ (p \wedge q))$
   Answer: We can first apply the following equality:

   $$\neg(\phi\ \mathbf{U}\ \psi)\ =\ \phi \wedge \neg\psi\ \mathbf{W}\ \neg phi \wedge \neg\psi$$

So, in our case we can rewrite our formula to:

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

Which can be simplified to:

$$p \wedge \neg q \ \mathbf{W} \ \neg p$$

Check the lecture notes how the automaton for **W** looks like :)

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

(c) $\Diamond\Box\Diamond\Box p$

Answer: Notice that $\Diamond\Box\Diamond\Box p$ can be simplified to $\Diamond\Box p$ using the Absorbtion law (see LN):

$\Diamond\Box\Diamond\phi \ = \ \Box\Diamond\phi$

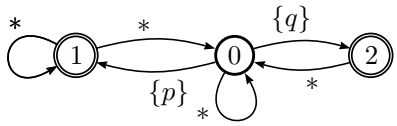
$\Box\Diamond\Box\phi \ = \ \Diamond\Box\phi$

You should be able to figure out how the automated for the resulting formula, $\Diamond\Box p$, looks like.

3. **LTL model checking** [3 pt].

(a) Consider the following generalized Buchi automaton K. The set of state predicates to consider is $Prop = \{p, q\}$. An arrow of the form $s \xrightarrow{*} t$ means that a transition from s to t is always possible with any label from 2^{Prop}.

K, 0 is the initial state:

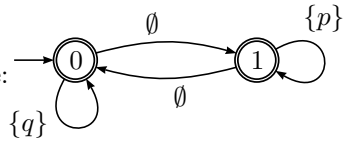


We have **two groups** of accepting states, namely: $F_1 = \{1\}$ and $F_2 = \{2\}$.

Give an equivalent **ordinary** (non-generalized) Buchi automaton that represents the same language. (Hint: which LTL property does the above automaton represent?)

(b) Below is an ordinary Buchi automaton, M, representing a program. The $Prop$ is the same as K above.

M, 0 is the initial state:



All states are acceptance states.

i. Construct the automaton that represents $M \cap K$.

ii. Let ϕ be the LTL formula represented by K. Is $\neg\phi$ a valid property of M? Explain your answer in terms of $M \cap K$.

4. **Model checking of programs with concrete states** [1 pt].

(a) In Promela, a system is made of a finite number of concurrent processes. Each process is sequential, and can be thought to maintain a program counter, whose value points to the next statement in the process to execute.

How can you *use LTL model checking* to check if a Promela system contains a 'dead' statement? A dead statement is a statement that will never be executed.

Answer: 0.4 pt.

We can assume from the above that every statement has a unique ID/address. Furthermore we can reasonably assume that there are only finitely many statements. For every valid statement ID l of process P, verify $\Box(P.pc \neq l)$. If it is **not** valid, there there is one execution that leads to $P.pc = l$, so the statement l will thus be executed in the execution. In other words, l is **not** dead. On the other hand, if the property is valid then statement l cannot be reached by any execution, hence dead.

Note that this is not a good check: $\Diamond(P.pc = l)$: this would check if *all* executions lead to s.

Arguably, we can also do the checking as follows. We can run a concrete state model checker a la Spin against the property *true*. Effectively, this will built the entire concrete state automaton representing the system. We modify the model checker to keep track which statement was used whenever it performs a step to extend the automaton it tries to build. Statements that were never invoked are thus dead. However, this is not not really the solution that is meant by this exercise as it does not really demonstrate the strength of LTL model checking. Will award less point for this.

(b) Below we describe a system consisting of 3 processes (in Promela notation), namely P, Q, Obs. The system operates on global variables x, y and a global channel c; x, y are of type integers and are initially 0, and c is a channel of size 0.

The process P keeps increasing the value of x, modulo 3. Q waits until x>0, and then sends the value of x over the channel c. Note that (x>0) in the code of Q is a blocking expression a la Promela.

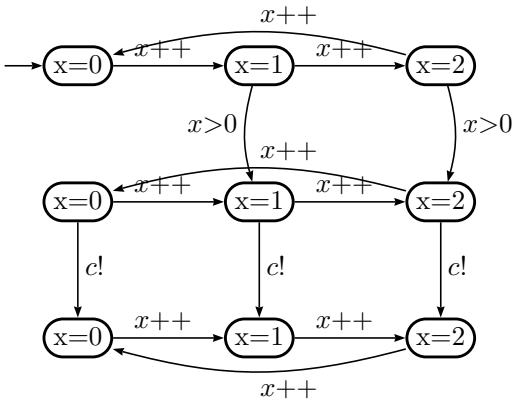
active proctype P {**do** :: $x = (x+1)$ **mod** 3 **od**}
active proctype Q {$(x>0)$; $c!x$}
active proctype Obs {$c?y$}

What are the possible values that Obs can receive from the channel c?

Answer: [0.2pt] 0,1,2

(c) Construct a concrete state finite state automaton representing $P||Q||Obs$.

Answer: 0.4 pt.



5. **CTL model checking** [1 pt].

Consider again the program represented by the Buchi automaton M in the question 3b. Consider the CTL property:

$$\phi \;=\; \mathbf{AF}(\mathbf{E}[p\ \mathbf{U}\ (\neg p \wedge \neg q)])$$

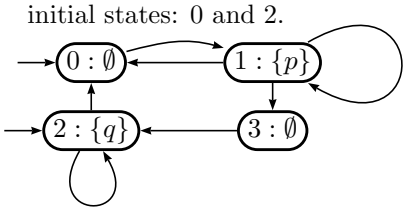
Give an algorithm to model check ϕ on M. (Note that M is not a Kripke structure)

Answer: First, convert the Buchi automaton in 3b to a Krikpe structure. It has to be as such that they define the same language. This can be done systematically by representing every arrow in the original Buchi automaton with its own state in the Kripke, labelled with the same label.

Then, in the Kripke we connect state s to t if in the Buchi, t is an arrow that immediately follows s.

The initial states of the Krike is all those states representing outgoing arrows from the Buchi's initial state.

The Kripke below is not obtained through the above procedure though; because I optimize it a bit to reduce the number of arrows. I add numbers to identify the states. We have two initial states: 0 and 2.



We recurse down the formula ϕ to label the above automaton with the sub-formulas of ϕ. The labelling works bottom-up. We label a state s with f iff $tree(s) \models f$.

(a) We start with the labeling with the sub-formula p. This formula only holds on the state 1. So, $W_p = \{1\}$.

(b) Next is the labelling with the sub-formula $\neg p \wedge \neg q$. This formula only holds on the state 0 and 3. So, $W_{\neg p \wedge \neg q} = \{0, 3\}$.

(c) Next, we do the labelling of the subformula $\psi = \mathbf{E}[p\ \mathbf{U}\ (\neg p \wedge \neg q)]$. We calculate W_ψ by iteratively calculating and improving approximation $Z_1, Z_2, ...$ we reach a fix point Z_k, which is then the W_ψ we look for.

i. In the first approximation, Z_1, we can immediately label all the states where $\neg p \wedge \neg q$. So, $Z_1 = W_{\neg p \wedge \neg q} = \{0, 3\}$.

ii. Z_2 is Z_1 plus all the states where p holds, and have at least one outgoing transition to Z_1. This results in the addition of state 1. So, $Z_2 = \{0, 1, 3\}$

iii. Z_3 is Z_2 plus all the states where p holds, and have at least one outgoing transition to Z_2. But this does not add any new state. So, Z_2 is our fix point.

(d) Finally, we now label with the top-formula ϕ. Again, we do this iteratively until we reach a fix point.

i. We start with $Y_1 = W_\psi = \{0, 1, 3\}$.

ii. Y_2 is Y_1 plus all states whose all outgoing transitions lead to Y_1. It turns out that no state can be added. So, Y_1 is the fix point.

(e) It turns out that while the formula ϕ holds in the initial state 0, it does not hold in the other initial state 2. So, it is not a valid property of the program.

6. **Symbolic model checking** [1 pt].

Consider a Kripke structure $M = (S, s_0, R, Prop, V)$ with $Prop = \{p, q\}$. S consists of 8 states. The transition relation R is symbolically represented by the Boolean formula below:

$$(\bar{x}.y')\ \vee\ (\bar{y}.z')\ \vee\ (\bar{z}.x')$$

We write for example $e_1.e_2$ as a shorthand for $e_1 \wedge e_2$. We write $\bar{e}$ to mean the negation $\neg e$.

Above, x, y, z are boolean variables representing the states in S. In the transition relation above, the primed version of these variable, x', y', z' represents the next-states of the transitions that the relation describes.

We assume the initial state $s_0 \in S$ to be represented by the formula $\bar{x}.\bar{y}.\bar{z}$.

In M, the function V describes the labeling of p and q on the states in S. This labelling is now encoded with the following Boolean formulas, for p and q respectively:

$$\begin{array}{lcl} W_p & = & x.z \vee y \\ W_q & = & \bar{y} \end{array}$$

Questions:

(a) Give a Boolean formula that represents the set of all successor states of the initial state s_0.

Answer: 0.4 pt. The initial state s_0 is represented by the formula $\bar{x}.\bar{y}.\bar{z}$. So, all its successors can be obtained by filling in this encoding to the transition relation:

$$(\bar{x}.y')\ \vee\ (\bar{y}.z')\ \vee\ (\bar{z}.x')[0, 0, 0/x, y, z]$$

So, we obtain:

$$(\bar{0}.y') \ \vee \ (\bar{0}.z') \ \vee \ (\bar{0}.x')$$

which can be simplified to $y' \vee z' \vee x'$, except not we need to turn the primed vars to non-primed vars. So

$$y \vee z \vee x$$

(b) How to check if $p \rightarrow q$ is a valid property of M?
Answer: 0.3 pt. $p \rightarrow q$ is equivalent to $\neg p \vee q$. The states satisfying this are those in $\neg W_p \vee W_q$; so:

$$W_{p \rightarrow q} \ = \ \neg(x.z \vee y) \vee \bar{y}$$

To check if $p \rightarrow q$ is valid on M, we check if $W_{p \rightarrow q}$ evaluates to true on M's initial state s_0; in other words on $x, y, z = 0, 0, 0$. Filling in these values the formula become:

$$\neg(0.0 \vee 0) \vee \bar{0}$$

which does evaluate to 1. So, $p \rightarrow q$ is valid on M.

(c) Is **EX** q a valid property of M? Explain.
Answer: 0.3 pt. The formula characterizing the states on which the formula hold is:

$$W_{\mathsf{EX}q} \ = \ (\exists x', y', z' :: \ R \wedge W_q[x', y', z'/x, y, z])$$

Filling in R and W_q we obtain:

$$W_{\mathsf{EX}q} \ = \ (\exists x', y', z' :: \ ((\bar{x}.y') \ \vee \ (\bar{y}.z') \ \vee \ (\bar{z}.x')) \ \wedge \ \bar{y'})$$

$\mathsf{EX}q$ is valid on M if $W_{\mathsf{EX}q}$ evaluates to true on s_0. Filling this is, the formula becomes:

$$(\exists x', y', z' :: \ ((\bar{0}.y') \ \vee \ (\bar{0}.z') \ \vee \ (\bar{0}.x')) \ \wedge \ \bar{y'})$$

which can be simplified to:

$$(\exists x', y', z' :: \ (y' \vee z' \vee x') \wedge \bar{y'})$$

This formula is a tautology. So yes, $\mathsf{EX}q$ holds on M's initial state, thus on M.

7. **BDD-based model checking** [0.5 pt].

(a) Explain how to use BDD to check if a given state t is reachable from the initial state of a program with finite number of states.
Answer: (0.3pt) We can covert this into a model checking problem with $Prop = \{p\}$, with the state t to be the only state in the program where p holds. Then t is reachable from the initial state if and only if the CTL formula $\mathsf{EF}p = \mathsf{E}[\mathsf{true} \ \mathsf{U} \ p]$ is valid on the program. This can be checked with CTL-model checking in the usual way. However, now we do this symbolically.
We represent the transition relation of the program with a Boolean function $R(x, x')$ where x and x' are boolean vectors. Let W_p be the Boolean function representing the states where p holds (which would then contain only the state t).
We iteratively calculate the states on which the formula $W_{\mathsf{EF}p}$ hold. We do this by calculating increasing approximation $Z_0, Z_1, ...$:

$$\begin{array}{lcl} Z_1 & = & W_p \\ Z_{i+1} & = & Z_i \vee (W_{\mathsf{true}} \wedge (\exists x' . \ R(x, x') \wedge Z_i[x'/x])) \end{array}$$

where W_{true} is a boolean formula representing the set of states in the program where true. Well, the latter would of course hold on any state. So, W_{true} is just true. So, the construction of Z_{i+1} can be simplified to:

$$Z_{i+1} \;=\; Z_i \vee (\exists x'.\, R(x, x') \wedge Z_i[x'/x])$$

We stop when we reach a fix point; this fix point would be the formula representing $W_{\mathsf{EF}p}$ that we look for.

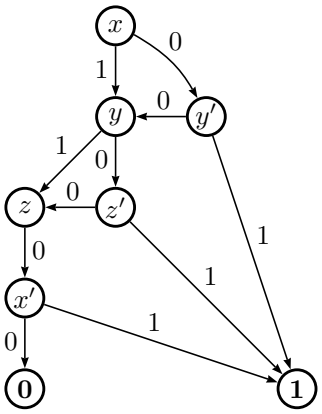
By using BDD to represent Z_i's, we have a means to efficiently check if we have reached a fix point. This amounts to checking that the BDDs of Z_i and Z_{i+1} are structurally the same.

Next, we check if $W_{\mathsf{EF}p}$ contains the initial state s_0. We can do this by checking if the formula $W_{\mathsf{EF}p}$ would evaluate to true when given the value assignment representing s_0.

(b) Represent the program M from the question No. 6 as an ordered and reduced BDD. Mention your ordering.

Answer: 0.2pt. The formula happens to be in the DNF form, which makes it easier to construct the corresponding reduced OBDD. See below. The chosen ordering O is of the variables is $x < y' < y < z' < z < x'$. This means that every path in the BDD should respect this ordering. That is, variables that appear in the path, from the root to the leaf, appear in an order that respects O.

You can choose a different ordering, and may end up with a different shape of your OBDD. Just keep in mind that some ordering may lead to a large BDD (e.g. if the ordering was $x < y < z < x' < y' < z'$).



8. **Model checking** [0.5 pt].

Let K and M be two **generalized** Buchi automata. They have indeed finite number of states. Their labels are taken from **the same** finite set of alphabets. We write $K \sqsupseteq M$ (K 'refines' M) to mean that all sentences accepted by M are also accepted by K. Propose a way to check this in finite number of steps.

Answer: We need to check $K \sqsupseteq M = L(M) \subseteq L(K)$. If K can be equivalently described by an LTL formula ϕ_K the problem is the same as checking whether ϕ_K is a valid LTL property of M. This proceeds by constructing the Buchi automaton representing $\neg\phi_K$, and the rest of the process is known to you.

This idea relies on the fact that you can construct ϕ_K. Both M and K are assumed to operate on the same alphabet, say A. Because LTL operates on 'state predicates' (as its atomic level formulas), from the LTL's perspective we will treat every symbol $p \in A$ and every state s in K as 'state predicates'.

We will construct ϕ_K as follows:

$$\phi_K \;=\; @s_0 \wedge R \wedge \mathcal{F}$$

For a state s_i, $@s_i$ means that the automaton K is in state s_i. This is encoded by the conjunction $\neg s_0 \wedge ... \wedge \neg s_{i-1} \wedge s_i \wedge \neg s_{i+1} \wedge ... \wedge \neg s_n$.

R encodes the arrows in K. For every state s in in K we add the following conjunct to R. It quantifies over all outgoing arrows to states $t_1...t_k$ labelled with $p_1...p_k$.

$$\Box(@s \rightarrow \mathsf{X}((p_1 \wedge @t_1) \vee ... \vee (p_k \wedge @t_k)))$$

$\mathcal{F}$ encodes K's accepting conditions. For every accepting group F in K, we add the following conjunct to $\mathcal{F}$. It quantifies over all members $u_1...u_n \in F$.

$$\Box\Diamond(@u_1 \vee ... \vee @u_n)$$

Now that we have an LTL formula representing K, we can construct its negation, and then construct a Buchi automaton representing its negation.

The automaton M will have to be adjusted a bit. Every transition labelled with some p in M should be treated that the transition does not care about the value of s_k predicates in ϕ_K.