Exam-2 Program Verification 2016/2017

BBG-209, 30th Jan. 2017, 13:30–16:30

Lecturer: Wishnu Prasetya

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

   Let P be a process, and r a resource that P may want to use. Formalize the following properties in LTL or CTL. You can introduce state predicates to abstractly capture needed concepts, e.g. $req(P, q)$ 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) The description of P mentions that that are some states that can be called 'idle states'. If P is in such state, we say that P is idle, and otherwise non-idle. Express this property: as long as P remains idle, it will not request access to the resource r.

   Answer: LTL: $\Box(idle \rightarrow \neg req(P, r))$

   (b) If P is idle, it should be *possible* for P to become non-idle. Note that we do not want to require it to eventually become non-idle. We do want to insist that there should be a possibility to become non-idle.

   Answer: CTL: $\mathbf{AG}(idle \rightarrow \mathbf{E}(true\ \mathbf{U}\ \neg idle))$

   (c) Whenever P requests to access the resource r, it will maintain the request, and eventually the request should be granted.

   Answer: LTL: $\Box(req(P, r) \rightarrow (req(P, r)\ \mathbf{U}\ granted(P, r)))$

   (d) When P is granted access to the resource r, it should be *possible* for P to later release r *without* in the mean time ever uses r.

   Answer: CTL: $\mathbf{AG}(granted(P, r) \rightarrow \mathbf{E}(\neg use(P, r)\ \mathbf{U}\ release(P, r)))$

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

   Give for each of the LTL formula below, a (generalized) Buchi automaton that equivalently describes the formula. For each automaton, specify which states are the initial states, and what are its groups of accepting states.

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

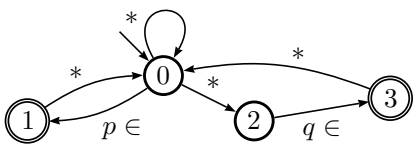
   Answer: Notice first that $\neg(p\ \mathbf{U}\ q) = (p \wedge \neg q)\ \mathbf{W}\ (\neg p \wedge \neg q)$. The latter is easier to imagine how to represent it as a Buchi automaton. Check the slides on how a Buchi automaton for $a\ \mathbf{W}\ b$ looks like :)

   (b) $\Box(p\ \mathbf{U}\ q)$

   Answer: Check the slides for a Buchi automaton of $p\ \mathbf{U}\ q$. Then, you only need to figure out how to do the outer $\Box$ part.

   (c) $(\Box\Diamond p)\ \wedge\ (\Box\Diamond \mathbf{X}\ q)$

   Answer:



The initial state is 0. The above is a *generalized* Buchi automaton with *two groups* of accepting states: $\{1\}$ and $\{3\}$. An execution is only accepting if it passes each group infinitely many often (though, if a group F has multiple accepting states, e.g. $F = \{u, v\}$, it is not required to pass each u as well as v infinitely many often. Passing one of them infinitely many often is good enough.).

Note for each generalized Buchi automaton there exists a regular Buchi automaton (which only has one group of accepting states.) that accepts the same language. Check the Lecture Notes.

(d) $\Diamond\Box\Diamond\Box p$

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

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

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

The resulting formula, $\Diamond\Box p$ ... is easy to figure out how to represent that as a Buchi automaton.

3. **Concepts related to FSA** [1 pt].

Let $M_1 = (S_1, i_1, F_1, R_1, \Sigma_1)$ be a Buchi automaton, where S_1 is its set of states, i_1 is its (single) initial state, F_1 is its set of accepting states, Σ_1 is the alphabet of the labels on M_1's transitions, and R_1 describes the transitions, such that if s is a state and $a \in \Sigma_1$, then $R_1(s, a)$ describes transitions from s labelled by a. There is a transition (arrow) labelled with a that goes from the state s to a state t if and only if $t \in R(s, a)$.

Analogously, let $M_2 = (S_2, i_2, F_2, R_2, \Sigma_2)$ be another Buchi automaton.

Give an algorithm to construct a Buchi automaton M_3, such that $L(M_3) = L(M_1) \cap L(M_2)$.

Answer:

The states: $S_3 = S_1 \times S_2$. The initial state is (i_1, i_2). We will only allow symbols which are allowed by both M_1 and M_2. So, $\Sigma_3 = \Sigma_1 \cap \Sigma_2$. It is easier to define M_3 as a generalized Buchi automaton, with:

$$\begin{array}{lcl} F_3 & = & \{A_1, A_2\} \\ A_1 & = & \{(f, t) \mid f \in F_1,\ t \in S_2\} \\ A_2 & = & \{(s, g) \mid s \in S_1,\ g \in F_2\} \end{array}$$

Notice that this choice of accepting groups will only accept executions that pass an accepting state of M_1 inifinitely many often **and** pass an accepting state of M_2 infinitely many often.

The transition relation R_3 is defined by:

$$R_3\ (s_1, s_2)\ \alpha \;=\; \{(t_1, t_2) \mid t_1 \in R_1\ s_1\ \alpha,\ t_2 \in R_2\ s_2\ \alpha\ \}$$

Notice that the above definition only allows a transition from $(s_1, s_2) \xrightarrow{\alpha} (t_1, t_2)$ if both the transition $s_1 \xrightarrow{\alpha} t_1$ is possible in M_1 and $s_2 \xrightarrow{\alpha} t_2$ is possible in M_2. Notice the common symbol α.

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

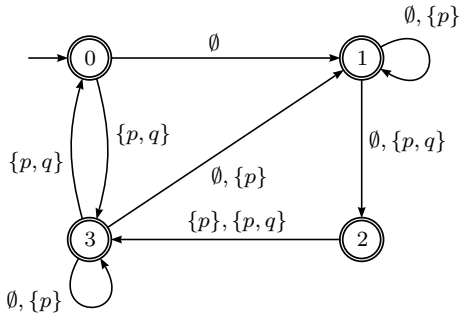
Consider a program M modeled by a Buchi automaton shown below (left). 0 is the initial state. All states are accepting. The set of state properties to consider is $Prop = \{p, q\}$. The arrows are explicit labelled by subsets of $Prop$.

If a transition from s to t has multiple labels, it means that we can go from s to t, through *any* of the labels.

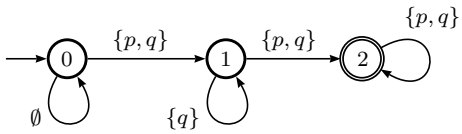
Consider an LTL property ϕ that we want to verify on the program. The Buchi automaton K below represents the formula $\neg\phi$. The initial state is 0, and only state 2 is accepting.

Both Buchi automata are ordinary Buchi automata (not generalized ones).

M :



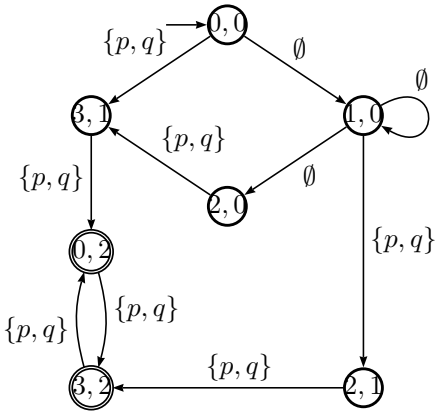
K :



(a) What is this ϕ? (express it in LTL; you can use $\neg$)
Answer: Well, $\neg\phi$ is:

$$\bar{p}\bar{q}\ \mathsf{U}\ (pq \wedge \mathsf{X}(\bar{p}q\ \mathsf{U}\ \Box pq))$$

(b) Construct $M \cap K$. Is ϕ valid? Explain your answer in terms of $M \cap K$.
Answer: The intersection automaton is shown below, with $(0,0)$ as the initial state and the states $(0,2)$ and $(3,2)$ as the accepting states. The language of this automaton is not empty. That is, it has a reachable accepting state f (namely the state $(0,2)$) and a cycle that passes through this f. Hence the original property ϕ is not valid.



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

Let a program M be modeled by a Kripke structure $(S, i_0, R, Prop, V)$, where S is M's set of states, i_0 is its single initial state, R is a function describing the transitions between the states, $Prop = \{p, q\}$ is a set of state properties (predicates), and V is a function that labels the states with members of $Prop$.

Consider a CTL property $\phi = \mathsf{E}(p\ \mathsf{U}\ \mathsf{A}(q\ \mathsf{U}\ (p \wedge q)))$. Give an algorithm to model check ϕ on M.

Answer:

(a) Let W_p and W_q be the states where respectively p and q hold. This can be directly derived from V. W_p consists of all the states s on which $p \in V(s)$; and analogously W_q.

(b) We first label the states $W_{p \wedge q}$ on which $p \wedge q$ hold. These are the states s for which $V(s) = \{p, q\}$.

(c) We label those states on which $\mathbf{A}(q \ \mathbf{U} \ p \wedge q)$ hold. Let's call the set of these states: $Z = W_{\mathbf{A}(q \ \mathbf{U} \ p \wedge q)}$. The labelling proceeds iteratively as follows. We approximate Z with a series of Z_k:

   i. We start with $Z_1 = W_{p \wedge q}$.

   ii. For subsequent $k \geq 1$, Z_{k+1} is Z_k plus all states s which are members of W_q and furthermore *all* its successors are members of Z_k.

   iii. We stop once we come to a fix point, where $Z_{k+1} = Z_k$. This is our solution for Z.

(d) We label those states on which $\phi = \mathbf{E}(p \ \mathbf{U} \ \mathbf{A}(q \ \mathbf{U} \ p \wedge q))$ hold. Let's call the set of these states: $Y = W_{\mathbf{E}(p \ \mathbf{U} \ \mathbf{A}(q \ \mathbf{U} \ p \wedge q))}$. The labelling proceeds iteratively as follows. We approximate Y with a series of Y_k:

   i. We start with $Y_1 = Z$, where Z is as defined in the previous step.

   ii. For subsequent $k \geq 1$, Y_{k+1} is Y_k plus all states s which are members of W_p and furthermore at least *one* of its successors is a member of Y_k.

   iii. We stop once we come to a fix point, where $Y_{k+1} = Y_k$. This is our solution for Y.

(e) The property ϕ is valid on M iff $i_0 \in Y$.

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

Consider a Kripke structure $M = (S, i_0, R, Prop, V)$ with $Prop = \{p, q\}$. S consists of 8 states. The transition relation R is symbolically represented by the Boolean formula below:

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

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 i_0 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) Is p a valid property of M? Explain.

Answer: To check if p is valid, we check if the initial state i_0 is a member of W_p. Now, W_p is represented symbolically. State i_0 corresponds to the valuation $v : [x \mapsto 0, y \mapsto 0, z \mapsto 0]$. We can check its membership by checking if W_p is satisfied by this v (that is, whether it yields 1). It is not.

(b) Is $\mathsf{AX}\ q$ a valid property of M? Explain.
Answer: The formula characterizing the states on which the formula hold is:

$$W_{\mathsf{AX}q} \ = \ (\forall x', y', z' :: \ R \rightarrow W_q[x', y', z'/x, y, z])$$

Filling in R and W_q we obtain:

$$(\forall x', y', z' :: \ ((x.\bar{y}.y'.z') \ \vee \ (\bar{x}.(x' \vee \bar{y'})) \ \vee \ (z.x'.y')) \ \rightarrow \ \bar{y'})$$

To check if $\mathsf{AX}\ q$ is valid on M we check whether i_0 is in $W_{\mathsf{AX}q}$. Again, this amounts to checking if $W_{\mathsf{AX}q}$ is satisfied by the valuation $v : [x \mapsto 0, y \mapsto 0, z \mapsto 0]$. With this valuation, the formula to check is:

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

Notice that this formula still has bounded variables. Note that $W_{\mathsf{AX}q}$ is satisfied by the valuation $v : [x \mapsto 0, y \mapsto 0, z \mapsto 0]$ if the above formula is valid. That is, if it yields 1 on all possible x', y', z'.
The above formula can be further simplified to:

$$(\forall x', y', z' :: \ x' \vee \bar{y'} \ \rightarrow \ \bar{y'})$$

which is *not* a valid formula. So, $\mathsf{AX}\ q$ is NOT valid on M

7. **CSP** [1 pt].
Consider the following CSP processes P_1 and P_2. The alphabet of both is $\{a, b\}$. The $\rightarrow$ operator is right associative; so $a \rightarrow b \rightarrow P$ means $a \rightarrow (a \rightarrow P)$.

$$P_1 \ = \ (a \rightarrow \mathsf{STOP}_{\{a,b\}}) \ \Box \ (b \rightarrow a \rightarrow P_1)$$

$$P_2 \ = \ b \rightarrow a \rightarrow a \rightarrow \ (\mathsf{STOP}_{\{a,b\}} \ \sqcap \ (a \rightarrow \mathsf{STOP}_{\{a,b\}}))$$

(a) Does $P_1 \sqsubseteq P_2$ hold under the trace semantic? Explain.
Answer: In the trace semantic, $P_1 \sqsubseteq P_2$ means $trace(P_1) \supseteq trace(P_2)$. Notice that $baaa$ is a trace of P_2 but not of P_1. So the refinement does not hold.

(b) Consider another process Q with alphabet $\{b, c\}$ defined by:

$$Q \ = \ (b \rightarrow c \rightarrow Q) \ \sqcap \ \mathsf{STOP}_{\{b,c\}}$$

Give an Finite State Automaton (FSA) that equivalently describes Q under the failures semantic. Label each state in your FSA with the *initials* and *refusals* of that state.

(c) Give $\mathsf{refusals}(P_1||Q)$.
Answer:

$$\begin{array}{lcl} refusals(P_1) & = & \{\emptyset\} \\ refusals(Q) & = & \{\emptyset, \{b\}, \{c\}, \{b, c\}\} \end{array}$$

$refusals(P_1||Q) = \{X \cup Y \mid X \in refusals(P_1), Y \in refusals(Q)\}$. Since P_1 only has $\emptyset$ as its only refusal, it follows that $refusals(P_1||Q)$ is just equal to the refusals of Q.

8. **LTL model checking** [0.5 pt].

   Suppose we allow a Promela model P to have infinite number of states. Indeed, we cannot verify such a model through model checking. However, we can still do testing. Suppose we want to have a set of test cases with 100% actions-coverage. That is, every atomic action in P should have been executed at least once during the executions of all these test cases. Note that the test cases only need to fulfill these requirement collectively, rather than individually.

   Propose an idea of how we can exploit LTL model checking to automatically generate such a set of test cases.

9. **CTL model checking** [0.5 pt].

   In the standard CTL, the atomic formulas are state predicates. Suppose we extend CTL by allowing LTL formulas to appear as CTL's atomic formulas. The meaning is formally defined as follows. Let $M = (S, i_0, R, Prop, V)$ be a Kripke structure. For simplicity it has a single initial state i_0. Let t be a computation tree of M, rooted in some state in S. The notation $\mathsf{root}(t)$ denotes this state. Let f be an LTL property. Its meaning in our CTL extension is as follows:

$$M, t \models f \quad \stackrel{def}{=} \quad M_{\mathsf{root}(t)} \models_{LTL} f$$

   where $M_u \models_{LTL} f$ means that f is valid in LTL, with respect to the Kripke structure M where we replace its initial state with u.

   Propose a modification to CTL model checking to model check formulas of the above described extended-CTL.