Exercises PV 08/09

Wishnu Prasetya

December 18, 2009

CTL Model Checking

1. Imagine a simple webshop where you can buy USB sticks. To make it simple, the shop only sell one kind of stick, and you can only buy one at a time. A typical interaction with the shop is (deliberately underspecified):

   (a) The user clicks on the button *buy* to buy a stick. She will then be presented with a page specifying the price of stick, and a (secured) form where she can fill in her creditcard data.

   (b) If the user agrees, she can click ok *ok*. The transaction is then confirmed. Otherwise she can *cancel*.

   (c) She can repeat the procedure to buy more sticks.

   (d) It is possible to click on *help* to get information on how to use the webshop.

   Unfortunately, we don't have a real implementation of this webshop, you will have to imagine one yourself.

   (a) Come up with a Kripke structure that abstractly models your imaginary webshop. There should of course be enough detail in your Kripke so that we are able to check some properties given later. We will assume the Donini et al's WAG modelling framework

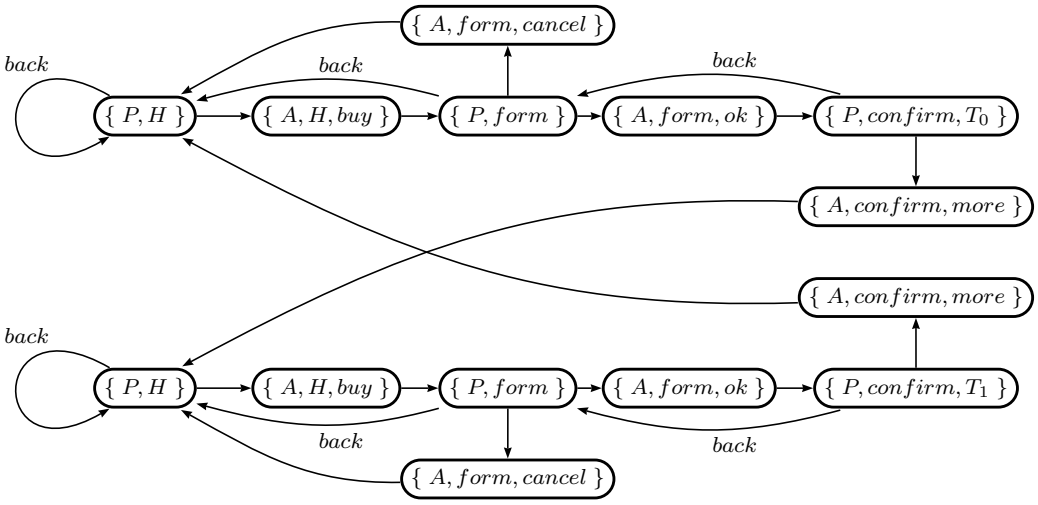
   Answer:

   In the WAG framework windows, pages, actions/links are represented by nodes. Let us introduce the propositions W, P, A to represent each sort respectively.

   Now, in my imaginary webshop, which is rather simple, every window has just one page. So, to make the model simpler I remove the W nodes from my model. The model is shown below.

   The edge labelled by *back* is the transition that will happen when the user click on her browser's back-button.

   In one property later there is a need to check on the number of completed transactions. Rather than explicitly counting transactions I will just abstractly distinguish one completed transaction with the next one. The propositions T_0 and T_1 are used for this purpose.



(b) The following are some properties that could be part of the webshop's specification. Express them with LTL or CTL.

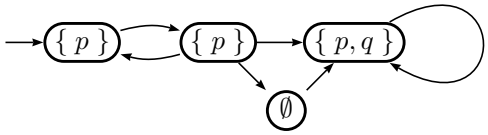
i. From any window, it should be possible to reach the help page.
Answer:
In LTL it may seems that this will capture the first property: $\Box(W \rightarrow \Diamond H)$, but this too strong. This formula would require that all paths starting from a window must eventually visit the help page. However, the property only requires that one path exists. This is sufficient, as it implies that it is possible for the user to navigate to the help page.
In CTL we can express this by $AG(W \rightarrow EF\ H)$.

ii. The user will not be charged double transactions if she accidentally clicks *ok* twice.
Answer:
This property is more difficult to capture. We can be express it indirectly by stating that it is not possible to get from T_0 to T_1 (and vice versa) without clicking on the *more* button. In CTL this is easy to express:

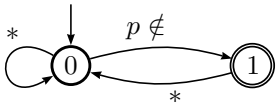
$$\mathbf{AG}\ (T_0 \rightarrow \neg\mathbf{E}(\neg more\ \mathbf{U}\ T_1))$$

(c) D escribe how CTL model checking works, then perform it to verify the above properties.

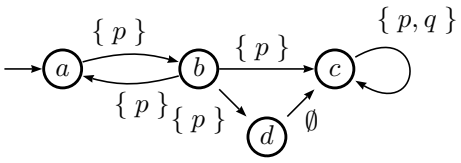
2. Consider the following Kripke structure:



(a) Do LTL model checking to verify the LTL property $\Diamond\Box p$.
Answer:
First we construct a Buchi automaton representing $\neg\Diamond\Box p \quad = \quad \Box\Diamond\neg p$:

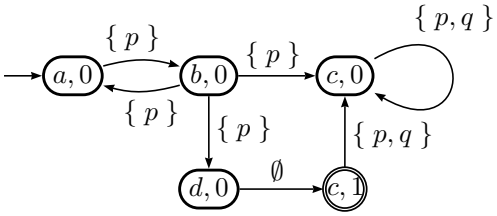


We transform the given Krike structure (modelling the program to verifty) to a Buchi automaton, because it would be easier as we later intersect it with the above specification:



We furthermore label the states with a, b, c, d.

We can now construct a Buchi automaton representing the intersection of the above two Buchis (belonging to the program and the negation of our original specification):



The given specification is violated if we can find a sentence accepted by the above Buchi automaton. This is a infinite sentence which can be produced by an infinite execution that pass through the accepting state above infinitely many times.

This is only the case if the accepting state is reachable (from the initial state), and if it is also part of a cycle. This can be detected e.g. by a nested DFS algorithm as used by SPIN.

In our case, the accepting state is not a part of any cycle. So, the automaton can't produce a counter example. Therefore, it satisfies $\Diamond\Box p$.

(b) Can the above property be expressed in CTL? How about in CTL*?

Answer: $\Diamond\Box p$ can't be expressed in CTL. We can express it in CTL* as: $AFG\ p$.

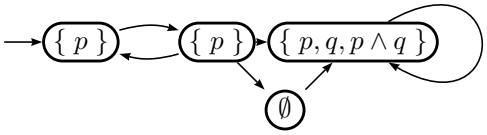
(c) Do CTL model checking to verify $EF(p \wedge q)$.

Answer:

The algorithm works by systematically labelling the states of our program with formulas known to hold on the corresponding states.

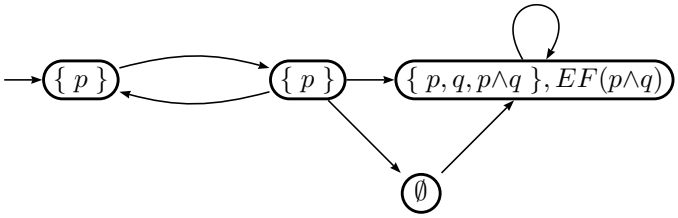
We first label the states with the atomic propositions that occur in our fomula. However, we don't have to do anything actually, because the labeling with the atomic propositions are already given in the Kripke itself.

We now continue with the labeling with the subformula $p \wedge q$:

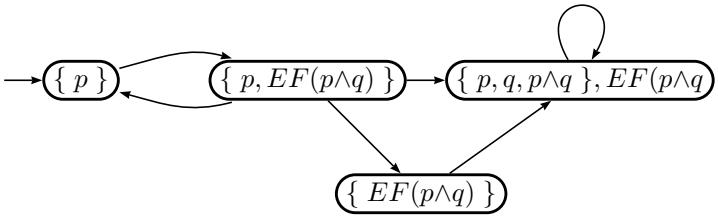


Then the labeling with the formula $EF(p \wedge q)$ itself. Notice that $EF(p \wedge q) = \mathit{true}\ \mathbf{U}\ (p \wedge q)$. The labeling of such a property is done iteratively.

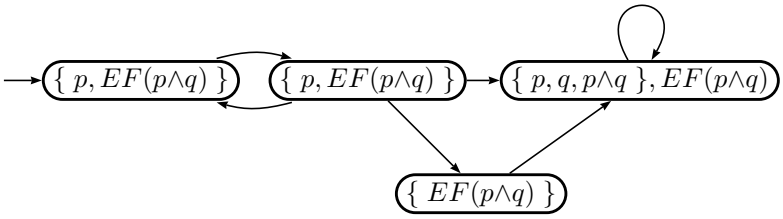
Iteration 1. Those states labelled with $p \wedge q$ obviously satisfy $EF(p \wedge q)$, so we add the latter:



Iteration 2. Next, all states s that has *an* outgoing transition into a state labelled with $EF(p \wedge q)$ will therefore satisfy $EF(p \wedge q)$; so we add it as labels:



Iteration 3. Applying the same step as in the previos iteration get us to this:



After this we can't label any more states with $EF(p \wedge q)$ (in this case simply because all states have been labelled with this formula). So we stop the iteration.

Note such an iteration will terminate. Each iteration either does not label any new state, in which case we stop, or it labels new states. The latter cannot go on forever because we have a finite number of states.

Now, since the initial state is also labelled with $EF(p \wedge q)$ the formula is therefor holds on this state; and thus the program satisfies it.

(d) Ok, now try these properties:

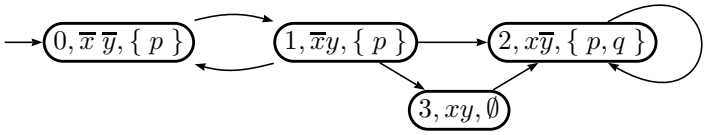
- $EF\ \neg p$
- $AG\ p$
- $E(p\ U\ (AG\ p))$
- $A(p\ U\ (AG\ p))$
- $AF AG\ p$

3. Consider again the Kripke structure in No. 2.

(a) How would you describe it if you are to express with a Boolean formula?

Answer:

The four states can be encoded by two boolean variables x, y. Let us first number the states and label them with their encoding. This is just so that we can latter refer to them. We will write $\overline{x}$ to mean $\neg x$.



We can now encode the arrows in this automaton with a boolean formula; each arrow can be encoded by one DNF clause (there are six arrows, notice that below we also have 6 clauses):

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

You'll of course get a formula with as many clauses as your arrows in the original automaton. However, we can come up with a more compact formula by expressing the automaton in a more declarative, rule-based, way. E.g.:

i. from 0, you may go to 1.

ii. from 1, you don't go back to 1.

iii. from 2 or 3, you may go to 2.

which can be directly expressed with boolean formulas, that now looks simpler:

$$
\begin{array}{l}
R(x, y, x', y') \\
= \\
\overline{x}\ \overline{y}\ \overline{x'}\ y' \ \vee\ \overline{x}\ y\ \overline{\overline{x'}\ y'} \ \vee\ x\ x'\ \overline{y'}
\end{array}
$$

The sets of states satisfying p respectively q are encoded by the formula:

$$
\begin{array}{rcl}
W_p & = & \overline{xy} \\
W_q & = & x\overline{y}
\end{array}
$$

(b) Do the model checking of the formula $EF(p \wedge q)$ on the symbolic representation of your Kripke.

Answer:

CTL symbolic model checking proceeds as CTL explicit state model checking, except that we use boolean formulas. When we label states with a formula, we basically compute the set of states satisfying that formula. We can express set of state with a boolean formula.

For example, in the next stage of labeling we would label all the states satisfying $p \wedge q$. These states are simply those states in the conjunction of W_p and W_q. So:

$$W_{p \wedge q} \ = \ W_p \wedge W_q \ = \ \overline{xy}x\overline{y} \ = \ x\overline{y}$$

Now we proceed with the labelling with $EF(p \wedge q)$. Remember we do this in iterations, where each iterations basically compute an approximation of the set of states satisfying $EF(p \wedge q)$.

The first approaximation K_0 is simply $W_{p \wedge q}$. The next approximation is:

$$
\begin{array}{rl}
K_1 & = \\
& (\exists x', y'.\ R(x, y, x', y') \ \wedge\ K_0[x', y'/x, y]) \ \vee\ K_0 \\
& = \\
& (\exists x', y'.\ R(x, y, x', y') \ \wedge\ x'\overline{y'}) \ \vee\ K_0 \\
& = \\
& (\exists x', y'.\ (\overline{x}\ \overline{y}\ \overline{x'}\ y' \ \vee\ \overline{x}\ y\ \overline{\overline{x'}\ y'} \ \vee\ x\ x'\ \overline{y'}) \ \wedge\ x'\overline{y'}) \ \vee\ K_0
\end{array}
$$

You will probably wonder if the above complicated formula really represent all the states labelled in the second iteration. It is indeed difficult to see this directly; but just so that you can convince yourself, let's simplify the above formula:

$$
\begin{array}{l}
\textit{last formula above} \\
= \\
\overline{x}\ y \ \vee\ x \ \vee\ K_0 \\
= \\
\overline{x}\ y \ \vee\ x \ \vee\ x\overline{y} \\
= \\
\overline{x}\ y \ \vee\ x
\end{array}
$$

The above formula encodes this set of states: $\{1, 2, 3\}$. If you recall the labelling in the explicit state procedure, this is indeed the set of states we label with $EF(p \wedge q)$ in Iteration-2.

The next approximation K_2 can be obtained in the same way:

$$
\begin{array}{rl}
K_2 & = \\
& (\exists x', y'.\ R(x, y, x', y') \ \wedge\ K_1[x', y'/x, y]) \ \vee\ K_1 \\
& = \\
& (\exists x', y'.\ R(x, y, x', y') \ \wedge\ (\overline{x'}\ y' \ \vee\ x')) \ \vee\ \overline{x}\ y \ \vee\ x
\end{array}
$$

At each iteration we should check if $K_{i+1} \leftrightarrow K_i$; if so we stop. I'm not going to write out K_2 above. I leave it to you to check yourself that we can stop after K_3, because $K_3 \leftrightarrow K_2$. Hence:

$$W_{EF(p \wedge q)} \;=\; K_3$$

Next we need to check if initial state is labelled by $EF(p \wedge q)$. In terms of symbolic model checking this means checking either of this:

i. Valuation $x{=}false$ and $y{=}false$ makes $W_{EF(p \wedge q)}$ true.

ii. $\overline{x}\,\overline{y} \;\rightarrow\; W_{EF(p \wedge q)}$ is valid.