Exercise Week 3: Symbolic Model Checking CTL

Jaco van de Pol

November 16, 2010

1 The Problem

Consider the following program, where x, y, z are boolean variables, and the guarded commands are executed non-deterministically:

$$
\begin{array}{l}
\textbf{do} \\
\quad \begin{array}{rcl}
\neg x & \rightarrow & x := 1 \\
x \wedge \neg y & \rightarrow & y := 1 \\
 & \rightarrow & z := \neg z
\end{array} \\
\textbf{od}
\end{array}
$$

Define the following properties on this system:

$$
\begin{array}{rcl}
\mathit{init} & :\equiv & \neg x \wedge \neg y \wedge \neg z \\
\mathit{error} & :\equiv & \neg x \wedge y \wedge \neg z \\
\mathit{pay} & :\equiv & y = \neg z \\
\mathit{goal} & :\equiv & x \wedge y \wedge z
\end{array}
$$

Question: Check with symbolic model checking in which states the following CTL properties hold:

- $\mathbf{AG}\,(\neg \mathit{error})$
- $\mathbf{E}[\neg \mathit{pay}\,\mathbf{U}\,\mathit{goal}]$
- $\mathbf{EG}\,y$

2 The solution

I will only work out the first example.
Step 1: Formalize the program's transition relation as a Boolean formula. Using a BDD package, this could be transformed to a BDD.
It is convenient to first formalize each command, and subsequently combine them by disjunction. This also gives us abbreviations that can be used later on. Written as a formula, with variables x, y, z (state before transition) and x', y', z' (state after transition) we get:

$$
\begin{array}{rcl}
\mathcal{R}_1 & :\equiv & \neg x \wedge x' \wedge y = y' \wedge z = z' \\
\mathcal{R}_2 & :\equiv & x \wedge \neg y \wedge y' \wedge x = x' \wedge z = z' \\
 & \equiv & x \wedge x' \wedge \neg y \wedge y' \wedge z = z' \\
\mathcal{R}_3 & :\equiv & x = x' \wedge y = y' \wedge z = \neg z' \\
\mathcal{R} & :\equiv & \mathcal{R}_1 \vee \mathcal{R}_2 \vee \mathcal{R}_3
\end{array}
$$

Step 2: rewrite the formula in the fragment EG, EU, EX.

$$
\begin{array}{rl}
 & \mathbf{AG}\,(\neg error) \\
\equiv & \neg\mathbf{EF}\,(\neg\neg error) \\
\equiv & \neg\mathbf{E}[True\,\mathbf{U}\,error]
\end{array}
$$

Step 3: now we compute formulas (computers would compute BDDs), representing the set of states that satisfy the subformulas. We do this bottom up.
Step 3a ($True$): this is easy, just the formula $True$ (or leaf 1 in BDDs). Note that this formula represents all 8 possible states.
Step 3b ($error$): this is also easy. The formula is just $\neg x \wedge y \wedge \neg z$, by definition. Note that this formula represents a unique state.
Step 3c ($\mathbf{E}[True\,\mathbf{U}\,error]$): All the work is in this step. For this EU formula we need to compute the least fixed point of a function (predicate transformer).

$$Lfp(Z \mapsto error \vee (True \wedge \mathbf{EX}\,Z))$$

Here EX is computed using the $Prev$ function, which is defined by:

$$Prev(\mathcal{S}, \mathcal{R}) :\equiv \exists \vec{x'}.\, \big(\mathcal{S}(\vec{x})[\vec{x'}/\vec{x}] \wedge \mathcal{R}(\vec{x}, \vec{x'})\big)$$

Extra explanation. In other words, we must compute the least fixed point of the function τ, defined by $\tau(Z) = error \vee Prev(Z, \mathcal{R})$. In order to do this, we frequently must compute $\exists \vec{v}.\, X \wedge \mathcal{R}$ for several X. Because $\mathcal{R}$ is biggish, we will often do this by using the following:

$$
\begin{array}{rl}
 & \exists \vec{v}.\, X \wedge \mathcal{R} \\
\equiv & \exists \vec{v}.\, X \wedge (\mathcal{R}_1 \vee \mathcal{R}_2 \vee \mathcal{R}_3) \\
\equiv & \exists \vec{v}.\, (X \wedge \mathcal{R}_1) \vee (X \wedge \mathcal{R}_2) \vee (X \wedge \mathcal{R}_3) \\
\equiv & (\exists \vec{v}.\, X \wedge \mathcal{R}_1) \vee (\exists \vec{v}. X \wedge \mathcal{R}_2) \vee (\exists \vec{v}. X \wedge \mathcal{R}_3)
\end{array}
$$

(actually, this corresponds to the idea of disjunctive partitioning from the lecture in week 2).
Another useful trick is the following: $\exists x.\, P \equiv P[0/x] \vee P[1/x]$, hence in particular:

$$\exists x.\, P \wedge x \wedge Q \equiv (P[0/x] \wedge 0 \wedge Q[0/x]) \vee (P[1/x] \wedge 1 \wedge Q[1/x]) \equiv P[1/x] \wedge Q[1/x]$$

And similarly,

$$\exists x.\, P \wedge \neg x \wedge Q \equiv P[0/x] \wedge Q[0/x]$$

In particular, if x doesn't occur in P and Q we can just drop it:

$$\exists x.\, P \wedge x \wedge Q \equiv \exists x.\, P \wedge \neg x \wedge Q \equiv P \wedge Q$$

Continue step 3c. So let us start. We must apply τ repeatedly, starting from the empty set. So we get:

$$B_0 \quad \equiv \quad \mathit{False}$$

Next, we compute:

$$\begin{array}{rcl} B_1 & \equiv & \mathit{error} \vee \mathit{Prev}(B_0, \mathcal{R}) \\ & \equiv & \mathit{error} \vee \exists \vec{x'}.\, \big(\mathit{False}[\vec{x'}/\vec{x}] \wedge \mathcal{R}(\vec{x}, \vec{x'})\big) \\ & \equiv & \mathit{error} \vee \exists \vec{x'}.\, \mathit{False} \\ & \equiv & \mathit{error} \vee \mathit{False} \\ & \equiv & \neg x \wedge y \wedge \neg z \end{array}$$

Next, for B_2 we must compute $\mathit{Prev}(B_1, \mathcal{R})$. As explained above, we do this in three steps:

$$\begin{array}{rcl} \mathit{Prev}(B_1, \mathcal{R}_1) & \equiv & \exists \vec{x'}.\, B_1(\vec{x})[\vec{x'}/\vec{x}] \wedge \mathcal{R}_1(\vec{x}, \vec{x'}) \\ & \equiv & \exists x', y', z'.\, (\neg x \wedge y \wedge \neg z)[x', y', z'/x, y, z] \wedge (\neg x \wedge x' \wedge y = y' \wedge z = z') \\ & \equiv & \exists x', y', z'.\, (\neg x \wedge y \wedge \neg z)[x', y', z'/x, y, z] \wedge (\neg x \wedge x' \wedge y = y' \wedge z = z') \\ & \equiv & \exists x', y', z'.\, (\neg x' \wedge y' \wedge \neg z') \wedge (\neg x \wedge x' \wedge y = y' \wedge z = z') \\ & \equiv & \exists x', y', z'.\, \mathit{False} \\ & \equiv & \mathit{False} \end{array}$$

So B_1 has no $\mathcal{R}_1$ predecessors. Similarly, one can check that $\mathit{Prev}(B_1, \mathcal{R}_2) \equiv \mathit{False}$. Finally, we compute:

$$\begin{array}{rcl} \mathit{Prev}(B_1, \mathcal{R}_3) & \equiv & \exists \vec{x'}.\, B_1(\vec{x})[\vec{x'}/\vec{x}] \wedge \mathcal{R}_3(\vec{x}, \vec{x'}) \\ & \equiv & \exists x', y', z'.\, (\neg x \wedge y \wedge \neg z)[x', y', z'/x, y, z] \wedge (x = x' \wedge y = y' \wedge z = \neg z') \\ & \equiv & \exists x', y', z'.\, (\neg x' \wedge y' \wedge \neg z') \wedge (x = x' \wedge y = y' \wedge z = \neg z') \\ & \equiv & \exists x', y', z'.\, \neg x \wedge \neg x' \wedge y \wedge y' \wedge z \wedge \neg z' \\ & \equiv & \neg x \wedge y \wedge z \end{array}$$

So,

$$\begin{array}{rcl} B_2 & \equiv & \mathit{error} \vee \mathit{Prev}(B_1, \mathcal{R}) \\ & \equiv & (\neg x \wedge y \wedge \neg z) \vee \mathit{False} \vee \mathit{False} \vee (\neg x \wedge y \wedge z) \\ & \equiv & \neg x \wedge y \end{array}$$

For the next iteration we check that $\mathit{Prev}(B_2, \mathcal{R}_1) = \mathit{False}$ and $\mathit{Prev}(B_2, \mathcal{R}_2) = \mathit{False}$, because $\neg x' \wedge y'$ contradict both $\mathcal{R}_1$ and $\mathcal{R}_2$.
Then we compute

$$\begin{array}{rcl} \mathit{Prev}(B_2, \mathcal{R}_3) & \equiv & \exists \vec{x'}.\, B_2(\vec{x})[\vec{x'}/\vec{x}] \wedge \mathcal{R}_3(\vec{x}, \vec{x'}) \\ & \equiv & \exists x', y', z'.\, (\neg x \wedge y)[x', y', z'/x, y, z] \wedge (x = x' \wedge y = y' \wedge z = \neg z') \\ & \equiv & \exists x', y', z'.\, (\neg x' \wedge y') \wedge (x = x' \wedge y = y' \wedge z = \neg z')) \\ & \equiv & \exists x', y', z'.\, (\neg x \wedge \neg x' \wedge y \wedge y' \wedge z = \neg z') \\ & \equiv & \exists z'.\, (\neg x \wedge y \wedge z = \neg z') \\ & \equiv & \neg x \wedge y \end{array}$$

Hence

$$B_3 \equiv error \vee Prev(B_2, \mathcal{R}_1) \equiv (\neg x \wedge y \wedge \neg z) \vee (\neg x \wedge y) \equiv \neg x \wedge y$$

Clearly, $B_2 \equiv B_3$, so this is the smallest fixed point, and represents the set of states where $\mathbf{E}[True\,\mathbf{U}\,error]$ holds.
Step 3d ($\neg\mathbf{E}[True\,\mathbf{U}\,error]$): This is easy again, we just negate the result of Step 3c, and obtain $\neg(\neg x \wedge y) \equiv x \vee \neg y$
Step 4, conclusion. The formula $\mathbf{AG}\,(\neg error)$ holds in all the states that satisfy $x \vee \neg y$, so in particular it holds in the initial state $(\neg x \wedge \neg y \wedge \neg z)$. So the program cannot enter the error state.