6.894 Lightweight Formal Methods

Lecture 14: Model Checking, Part II

Daniel Jackson, April 6, 05

LTL Model checking.

We've seen how CTL is checked. How about LTL?
The conventional method is based on automata: you construct a machine based on the LTL formula (called a Buchi automaton) that accepts paths that violate the formula; you compose it with the machine from the model being checked, and then you explore the composed machine to find a path to an accepting state that corresponds to a counterexample.

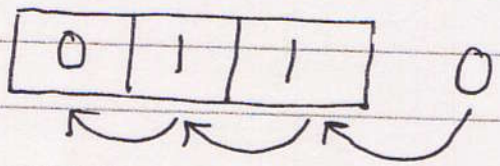
In this lecture though, we'll look at a different and newer method based on SAT. My reasons for choosing this are:

- it's an interesting method;
- it's the one used in NuSMV, the tool you'll use in your assignment;
- it's essentially what Alloy does when you set up a trace-based analysis.

This approach is called BOUNDED model checking, because (like Alloy, and unlike automaton based LTL methods) it considers only paths up to a certain length.

The intuition: Unrolling the Transition Relation

Take a 3-bit shift register:

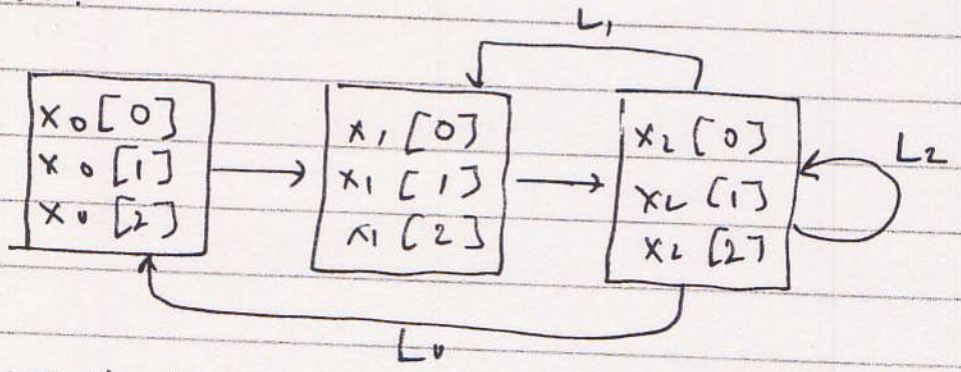


In each step, bits move to the left. Last bit gets 0.
So in 3 steps, should all be 0.

Suppose we have this model:

$$x'[0] = x[1]$$
$$\wedge\ x'[1] = x[2]$$
$$\wedge\ x'[2] = 1 \qquad \leftarrow \text{mistake!}$$

To analyze this, we'll consider paths of some bounded length. Say we consider paths of length $k = 2$, which requires $k+1 = 3$ states. We unroll the transition relation:



if we are checking

$\Diamond\ \forall i.\ x[i] = 0$

we need a path that satisfies

$\Box\ \forall \exists i.\ x[i] \neq 0$, which requires a Loop.

So one of the edges L_i' is needed to form the loop.

The formula we need to solve is:

initial condition just true — second step

$$I(x_0) \wedge T(x_0, x_1) \wedge T(x_1, x_2)$$

$$\wedge \quad V_i L_i$$

$$\wedge \quad \bigwedge_i S_i$$

where L_i' is the condition for the backedge

$$L_i = T(x_2, x_i)$$

and S_i is the condition that the register never be zero

$$S_i = x_i[0] \neq 0 \vee x_i[1] \neq 0 \vee x_i[2] \neq 0$$

In this case, we have a satisfying assignment by setting all $x_i[j]$ to $\underline{1}$.

Advantages of Bounded Model checking

- Because SAT uses DFS, counterexamples are found very quickly.
- No canonical representation needed; by avoiding BDDs, the problems of finding a var ordering, using exponential space, etc, go away.

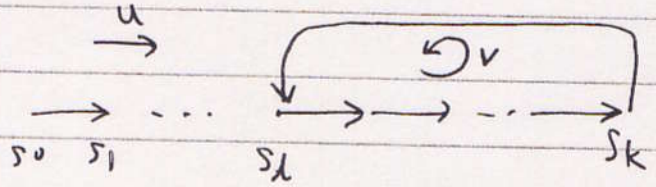
Disadvantages:

- It's bounded!
- Doesn't scale well to long paths.
- Some systems have long initialization sequences.

Looping Paths

Key idea is that paths can represent an infinite path if it's looping.

Say it has a (k, l)-loop if there's a back edge from k to l:



represent $u \cdot v^\omega$

Transition.

First we unfold the transition relation:

$$[\![M]\!]_k = I(s_0) \wedge \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1})$$

Then we translate the LTL formula.
There are 2 translations one for loops and one for loopless paths:

Loopless: (i = position along path)

$$[\![p]\!]_k^i = p(s_i')$$
$$[\![f \vee g]\!]_k^i = [\![f]\!]_k^i \vee [\![g]\!]_k^i$$

$$[\![\Diamond f]\!]_k^i = \bigvee_{j=i}^{k} [\![f]\!]_k^j$$
$$[\![\Box f]\!]_k^i = \text{false}$$
$$[\![\circ f]\!]_k^i = \text{if } i < k \text{ then } [\![f]\!]_k^{i+1} \text{ else false}$$

With a loop from k to l:

$$ {}_l[\![p]\!]^i_k = p(s_i) $$

$$ {}_l[\![f \vee g]\!]^i_k = {}_l[\![f]\!]^i_k \vee {}_l[\![g]\!]^i_k $$

} as before

$$ {}_l[\![\Diamond f]\!]^i_k = \bigvee_{j=\min(i,l)}^{k} {}_l[\![f]\!]^j_k $$

$$ {}_l[\![\Box f]\!]^i_k = \bigwedge_{j=\min(i,l)}^{k} {}_l[\![f]\!]^j_k $$

$$ {}_l[\![\circ f]\!]^i_k = {}_l[\![f]\!]^{succ(i)}_k $$

where $succ(i) = \begin{cases} i+1 & \text{if } i < k \\ l & \text{if } i = k \end{cases}$

Now we have the loop condition:

$$ {}_lL_k = T(s_k, s_l) $$

$$ L_k = \bigvee_{l=0}^{k} {}_lL_k $$

and putting it all together:

$$ [\![M, f]\!]_k = [\![M]\!]_k \wedge \left[(\neg L_k \wedge [\![f]\!]^0_k) \left(\vee \bigvee_{l=0}^{k} ({}_lL_k \wedge {}_l[\![f]\!]^0_k) \right) \right] $$

SAT formula for counterexamples to LTL formula f over model M considering paths of length $k-1$