Model Checking the Branching Time Temporal Logic CTL

Keijo Heljanko

April 20, 1998

1 Using Model Checking to Verify Concurrent Systems

Model checking can be used to check whether a concurrent system P satisfies a property ϕ we want it to have.

The system P is usually modeled using some kind of specification language: Petri nets, process algebra, automata, SDL, VHDL etc.

The specification ϕ is given using some temporal logic: CTL, LTL, CTL^*, μ-calculus.

The problem can be stated in two ways:

1. Does the program P meet it's specification ϕ?

2. Given a system P, and formula ϕ, is the behaviour of the system M a model of the formula ϕ?

The behaviour of the system M?

2 Generating the Kripke Model M

The Kripke model M can be obtained using reachability analysis of the system P behaviour.

How does one get the behaviour of the system P, i.e. the Kripke structure M?

- The Kripke structure M is given as input to the model checker.
- The Kripke structure M is generated from the system description P during model checking.
  - This is called **on-the-fly** model checking.

In on-the-fly model checking the structure M is generated during model checking. This has the nice property that it is not always necessary to generate the full structure M to decide whether M is a model of ϕ. This can result is big time and space savings in many cases.

On-the-fly model checking makes the model checking algorithm more difficult to implement and causes some overhead, so it's not always a win.

3 Global and Local Model Checking

There are two slightly different model checking problems:

- **Local model checking**: Is the formula ϕ true at a specified state s_0 of the system i.e. $M, s_0 \models \phi$?
- **Global model checking**: For which states of the Kripke structure M is ϕ true .i.e. what is the set of states $\{s | s \in S \wedge M, s \models \phi\}$?

Local model checking can be implemented together with on-the-fly model checking to create a very goal oriented model checker.

Global model checking can be made faster and more memory efficient in the worst case memory requirements sense. It is also much more straightforward to implement.

4 A Small Example

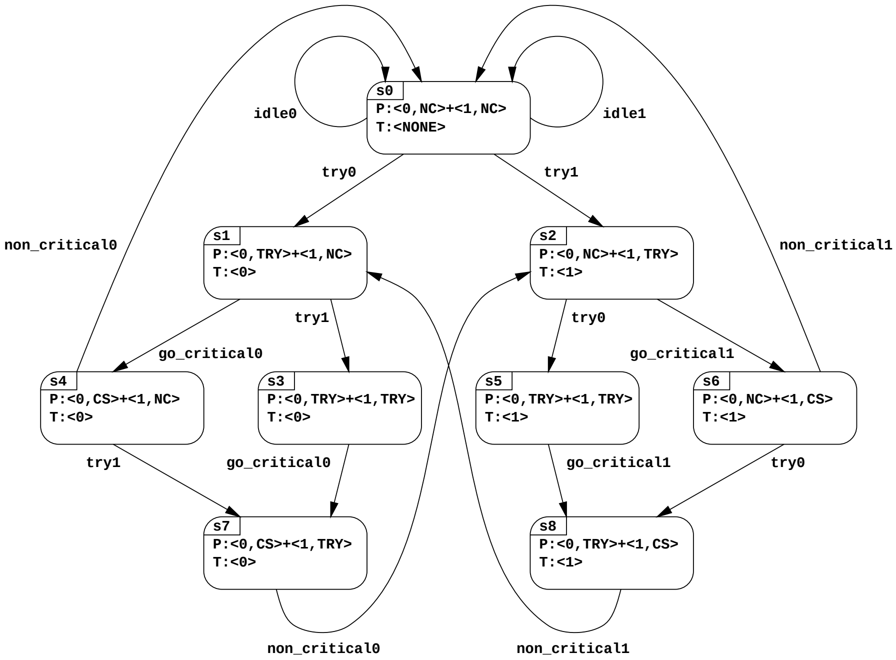

Figure 1: Reachability Graph of a Mutex System.

1. The processes are never at the same time in the critical section:

$$M, s_0 \models AG(\neg(P >= \ \langle 0, CS\rangle \wedge \ P >= \ \langle 1, CS\rangle))$$

2. If process 0 tries to get into the critical section, it will always eventually get there:

$$M, s_0 \models AG(P >= \langle 0, TRY\rangle \Rightarrow AF(P >= \langle 0, CS\rangle))$$

5 CTL Model checking

5.1 The Subformula Evaluation Order

We need to define in which order the proper subformulas are to be evaluated when evaluating a CTL formula f. Suppose we need to model check the CTL formula $f = A\,[p\,U\,E\,[q\,U\,\neg p]]$. The set of subformulas of f, $sub(f) = \{p, q, \neg p, E\,[q\,U\,\neg p]\,, A\,[p\,U\,E\,[q\,U\,\neg p]]\}$

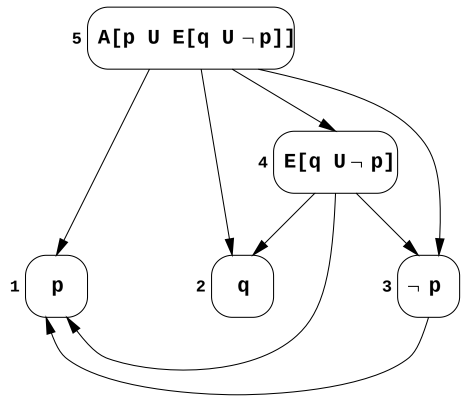

Figure 2: Subformula dependencies.

The basic idea is to model check all subformulas in all states in such an order that when evaluating a subformula i, all formulas on which it depends have been evaluated in all states. This order can be found with simple topological sorting.

There are several algorithms for model checking CTL, and four of them are presented in the summary below.

The time complexity for model checking CTL is only $O(|f| \times (|S| + |R|))$. All of the algorithms need $O(|f| \times |S| + |R|)$ space, under the assumption that all manipulated data items need $O(1)$ space each.

A table of the worst-case behaviour of the algorithms is below:

Algorithm	Time	Counter-examples
Naive	$O(\|f\| \times \|S\| \times (\|S\| + \|R\|))$	No
EMC	$O(\|f\| \times (\|S\| + \|R\|))$	No
ALMC	$O(\|f\| \times (\|S\| + \|R\|))$	Yes
New	$O(\|f\| \times (\|S\| + \|R\|))$	Yes

Figure 3: Worst-case time requirements.

6 Bibliography

References

[1] Clarke, E. M., Emerson, E. A., and Sistla, A. P.: **Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications**. ACM Transactions on Programming Languages and Systems 8 (1986) 2, pp. 244–263.

[2] Heljanko, K.: **Model Checking the Branching Time Temporal Logic CTL**. Research Report A45, Digital Systems Laboratory, Helsinki University of Technology, Espoo, 1997.

[3] Vergauwen, B., Lewi, J.: **A Linear Local Model Checking Algorithm for CTL**. In Best, E. (ed.): Proceedings of 4th International Conference on Concurrency Theory, Lecture Notes in Computer Science 715, Springer-Verlag, Berlin, 1993, pp. 447–461.