Symbolic Model Checking*

E. Clarke[1], K. McMillan[2], S. Campos[1] and V. Hartonas-Garmhausen[1]

[1] Carnegie Mellon University School of Computer Science, Pittsburgh, USA.
[2] Cadence Labs, Berkeley, USA

1 Introduction

Extensive simulation is currently the most widely used verification technique. However, simulation does not check all possible behaviors of a computing system. Exhaustive simulation is too expensive, and non-exhaustive simulation can miss important events, especially if the number of states in the system being verified is large. Other approaches for verification include theorem provers, term rewriting systems and proof checkers. These techniques, however, are usually very time consuming and require significant user intervention. Such characteristics limit the size of the systems they can verify in practice.

Temporal logic model checking [6, 7] is an alternative approach that has achieved significant results recently. Efficient algorithms are able to verify properties of realistic complex systems. In this technique, specifications are written as formulas in a propositional temporal logic and computer systems are represented by state-transition graphs. Verification is accomplished by an efficient breadth first search procedure that views the transition system as a model for the logic, and determines if the specifications are satisfied by that model.

Recent model checkers use symbolic algorithms, which allow the verification of extremely large state-spaces. In this approach the transition relation is represented implicitly by boolean formulas, and implemented by *binary decision diagrams* [1]. This usually results in a much smaller representation for the transition relation [16], allowing the size of the models being verified to increase up to more than 10^{20} states [2].

There are several other advantages to this approach. An important one is that the procedure is completely automatic. The model checker accepts a model description, specifications to be verified and determines, without user intervention, if the formulas are true or not for that model. Another advantage is that, if the formula is not true, the model checker will provide a counterexample. The counterexample is an execution trace that shows why the formula is not true. This is an extremely useful feature because it can help locate the source of the error and speed up the debugging process. Another advantage is the ability to verify partially specified systems. If a component hasn't been fully specified, some of its outputs can be assigned nondeterministic values. The set of

* This research was sponsored in part by the National Science Foundation under grant no. CCR-8722633, by the Semiconductor Research Corporation under contract 92-DJ-294, and by The Defense Advanced Research Projects Agency, Information Science and Technology Office, under the title "Research on Parallel Computing", ARPA Order No. 7330, issued by DARPA/CMO under Contract MDA972-90-C-0035.

behaviors modeled this way is a superset of the actual behaviors of the component. Useful information about the correctness of the system can be gathered before all the details have been determined. The abstracted model is then refined when more information about the component becomes available. This allows the verification of a system to proceed concurrently with its design. Consequently verification can provide valuable hints that will help designers eliminate errors earlier and define better systems.

The model checker used in this work is *Symbolic Model Verifier* (SMV) [16]. It has been applied successfully in the verification of several industrial systems. Examples include the Futurebus+ cache coherence protocol [9], the PCI Local Bus [3], a railway signalling system [14], an aircraft controller [5], a manufacturing system [13], and a medical monitoring system [4]. A survey about the technique can be found in [11].

2 Describing the System

The system being verified is described in the SMV language. We can specify synchronous or asynchronous, detailed deterministic or abstract nondeterministic finite state machines. The language provides modular hierarchical descriptions, reuse of components, and parameterization so that multiple instances of a module can use different data values. Within every module, local variables may be declared. The type of a variable may be boolean, an enumeration type or an integer subrange. For example:

```
VAR state0: {noncritical, trying, critical};
```

The value of the variables in each state are defined using `init` and `next`:

```
init(state0) := noncritical;
next(state0) :=
case
  (state0 = noncritical) : {trying,noncritical};
  (state0 = trying) & (state1 = noncritical): critical;
  (state0 = trying) & (state1 = trying) & (turn = turn0):
                          critical;
  (state0 = critical) :    {critical,noncritical};
  1: state0;
esac;
```

An SMV program can be viewed as a system of simultaneous equations whose solution determines the next state. When describing communication protocols, asychronous circuits, or other systems whose actions are not sychronized, we can define a set of parallel processes whose actions are interleaved arbitrarily in the execution of the program.

Fairness constraints can also be specified in SMV. A fairness constraint is an arbitrary set of states in the model, described by a temporal logic formula. A path in the model is considered fair with respect to a set of fairness constraints if each constraint is true infinitely often along the path (i.e. some state in the fair set of states is visited infinitely often). In SMV verification can be restricted to fair paths.

3 Verifying the System

Computation tree logic, CTL, is the logic used by SMV to express properties that will be verified. Formulas in CTL are built from atomic propositions, where each proposition corresponds to a variable in the model, boolean conectives $\neg$ and $\wedge$, and *temporal operators*. Each operator consists of two parts: a path quantifier followed by a temporal operator. Path quantifiers indicate that the property should be true of *all* paths from a given state (**A**), or *some* path from a given state (**E**). The temporal modality describes how events should be ordered with respect to time for a path specified by the path quantifier. They have the following informal meanings:

- **F** p — p holds sometime in the future.
- **G** p — p holds globally on the path.
- **X** p — p holds in the next state.
- p **U** q — q holds in the future, and p holds in all states until the state in which $q$holds.

The most common CTL operators are: **AG** p — p is globally true in all paths from the current state, i.e., p is *invariant*; **AF** p — p holds sometime in the future in all paths, i.e., p is *inevitable*; **EF** p — p holds sometime in the future for some path, i.e., p is *reachable*. Some examples of CTL formulas are given below to illustrate the expressiveness of the logic.

- **AG**$(req \rightarrow$ **AF** $ack)$: It is always the case that if the signal req is high, then eventually ack will also be high.
- **EF**$(started \wedge \neg ready)$: It is possible to get to a state where $started$ holds but $ready$ does not hold.
- **AG EF** $restart$: From any state it is possible to get to the $restart$ state.
- **AG**$(send \rightarrow$ **A**$[send$ **U** $recv])$: It is always the case that if $send$ occurs, then eventually $recv$ is true, and until that time, $send$ must remain true.

4 Conclusions

Symbolic model checking is a powerful formal specification and verification method that has been applied successfully in several industrial designs. Using symbolic model checking techniques it is possible to verify industrial-size finite state systems. State spaces with up to 10^{30} states can be exhaustively searched in minutes. Models with more than 10^{120} states have been verified using special techniques.

Several extensions to the original technique have been developed, making it even more powerful. Timing properties can be verified by performing a quantitative timing analysis [3, 5]. The designer can then analyze the performance of a system and gain insight in how well a system works early in the design process. Word-level model checking allows the verification of datapaths in addition to control [12]. Symmetry [8], abstraction [10, 15] and compositional reasoning [15] techniques significantly extend the power of model checking by exploiting the hierarchical structure of complex circuit designs and protocols.

More information about SMV, as well as the source code for the model checker can be found at: `http://www.cs.cmu.edu/~modelcheck`

References

1. R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8), 1986.
2. J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Symposium on Logic in Computer Science*, 1990.
3. S. Campos, E. Clarke, W. Marrero, and M. Minea. Verifying the performance of the PCI local bus using symbolic techniques. In *International Conference on Computer Design*, 1995.
4. S. V. Campos, E. M. Clarke, W. Marrero, and M. Minea. Timing analysis of industrial real-time systems. In *Workshop on Industrial-strength Formal specification Techniques*, 1995.
5. S. V. Campos, E. M. Clarke, W. Marrero, M. Minea, and H. Hiraishi. Computing quantitative characteristics of finite-state real-time systems. In *IEEE Real-Time Systems Symposium*, 1994.
6. E. M. Clarke and E. A. Emerson. Synthesis of synchronization skeletons for branching time temporal logic. In *Logic of Programs: Workshop, Yorktown Heights, NY, May 1981*. Springer-Verlag, 1981. *Lecture Notes in Computer Science*, volume 131.
7. E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.
8. E. M. Clarke, T. Filkorn, and S. Jha. Exploiting symmetry in temporal logic model checking. In *Proceedings of the Fifth Workshop on Computer-Aided Verification*, June 1994.
9. E. M. Clarke, O. Grumberg, H. Hiraishi, S. Jha, D. E. Long, K. L. McMillan, and L. A. Ness. Verification of the Futurebus+ cache coherence protocol. In L. Claesen, editor, *International Symposium on Computer Hardware Description Languages and their Applications*. North-Holland, April 1993.
10. E. M. Clarke, O. Grumberg, and D. E. Long. Model checking and abstraction. In *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Programming Languages*, January 1992.
11. E. M. Clarke, O. Grumberg, and D. E. Long. Verification tools for finite-state concurrent systems. In *A Decade of Concurrency — Reflections and Perspectives*, 1994. Springer Lecture Notes in Computer Science, 803.
12. E. M. Clarke, M. Khaira, and X. Zhao. Word level model checking — avoiding the pentium FDIV error. In *Design Automation Conference*, June 1996.
13. V. Hartonas-Garmhausen, E.M. Clarke, and S. Campos. Deadlock prevention in flexible manufacturing systems using symbolic model checking. In *International Conference on Robotics and Automation*, 1996.
14. V. Hartonas-Garmhausen, T. Kurfess, E.M. Clarke, and D. Long. Automatic verification of industrial designs. In *Workshop on Industrial-strength Formal specification Techniques*, 1995.
15. D. E. Long. *Model checking, abstraction and compositional reasoning*. PhD thesis, SCS, Carnegie Mellon University, 1993.
16. K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.