μcke – Efficient μ-Calculus Model Checking

Armin Biere[1]
armin@ira.uka.de, Institut für Logik, Komplexität und Deduktionssysteme,
Universität Karlsruhe, Am Fasanengarten 5, D-76128 Karlsruhe, Germany

Abstract. In this paper we present an overview of the verification tool μcke. It is an implementation of a BDD-based μ-calculus model checker and uses several optimization techniques that are lifted from special purpose model checkers to the μ-calculus. This gives the user more expressibility without loosing efficiency.

Introduction

In [5] μ-calculus model checking with BDDs has been proposed as a general framework for various verification problems like model checking of LTL and CTL or testing for bisimulation equivalence and language containment. With a μ-calculus model checker all these verification tasks could be handled with one tool. Also some applications of symbolic model checking [16] need the μ-calculus as a specification language. On the other hand the most successful applications of model checking [7,2,15,10] all used a model checker with a less expressive specification language than the μ-calculus. The reason for this restriction was that for special purpose specification languages *optimized* model checkers can easily be build [5].

For example the SMV system of McMillan [14] uses fixed allocations of BDD variables for μ-calculus variables (ordering of BDD variables) for current and next state variables and specialized algorithms (`collapse`) for the computation of the set of states reachable in one step from a given set of states.

Other optimizations [4,14] that avoid the construction of the global transition relation (incremental transition relation generation, partitioning, MBFS) or speed up the computation (forward analysis, frontier set simplification) were only presented for state space analysis or CTL model checking.

In [3] we have shown that all these optimizations can be lifted to the μ-calculus. Especially an automatic allocation algorithm for BDD variables is given. It operates on allocation constraints to generate an allocation that respects the heuristic that all substitutions needed for the evaluation of a μ-calculus term should be fast (fast substitutions do not change the structure of a BDD but only change the variable markings). This is a generalization of the annotation mechanism of [11].

We also presented the $_{\mathsf{compose}}\mathsf{ite}_{\exists}$ algorithm that is a generalization of the BDD algorithm `collapse` of the SMV system and of the preImg-Operator of [8]. It performs a substitution, the calculation of "$\mathsf{if}\cdot\mathsf{then}\cdot\mathsf{else}$" and a quantification in one pass and thus avoids the unnecessary construction of intermediate results.

For the evaluation of these methods we implemented the μ-calculus model checker μcke. The main goal was to construct a μ-calculus model checker that is as efficient as

[1] sup. by DFG GRK 209/2-96 "Graduiertenkolleg Beherrschbarkeit Komplexer Systeme"

special purpose model checkers like the SMV system and also easy to use. In addition it should be more expressive and more flexible.

Although we found some properties that are more naturally described with the μ-calculus than f. e. with CTL, the μ-calculus is in general not very comprehensible and should not be used as an interface for an engineer that is involved in the design of a system to be verified. A *front end* that translates the formal specification produced by the engineer into the μ-calculus should be used instead. This front end must be changed for different optimizations and different application domains. To ease these adaptions the input language of μcke is similar to C (C++), a widespread used programming language.

Another point is that for special purpose model checkers there exist algorithms for the generation of counter examples if the verification fails. Here we used the method of [13] for the construction of counter examples for the whole μ-calculus.

The author is aware of three further implementations of μ-calculus model checkers [11,12,18] based on decision diagrams. The first and third implementation do not use automatic allocation algorithms and the user has to provide the allocation himself. The system of Janssen [12] (used in [17]) uses dynamic variable reordering [19] instead. We used this approach in a first prototype of μcke too and we were not able to achieve an equally high performance as the SMV system. The reason for the problems with this approach is that the BDD variables allocated for μ-calculus variables bounded by quantifiers may also be reordered by dynamic reordering. So there is no way to enforce fast substitutions. Some simple verification problems like the calculation of the set of reachable states of an *n*-bit counter and a simple arbiter suggested that μcke is 6 to 9 times faster than [12].

μcke

In this section we give an example of the input language of the μcke model checker and show how the optimization of forward analysis can be formulated in the μ-calculus. The example is a version of the alternating bit protocol [6,1] with an explicit description of the channels between sender and receiver. The control state of the sender is an enumeration type and is the first part of the total state of the sender. It also has an alternating bit and needs a place to store the data for retransmission if the first transmission failed.

```
class StateOfSender {
  ControlStateOfSender state; bool ab; Data data; };
```

The states of the channels and the receiver are defined in the same way. The global state of the system consists of the states of the sender, the receiver and the states of the two channels and of a running variable used to model the interleaving semantic:

```
class State {
  Running running; S2RChannel s2r; R2SChannel r2s;
  StateOfSender sender; StateOfReceiver receiver; };
```

Now we define the transition relation of the Sender and the global transition relation with a syntax similar to the definition of a function in C without curly parentheses enclosing the body:

```
bool TransSender(State s, State t)
  s.running = sender & CoStabSender(s,t) & (
    case
      s.sender.state = get :
        t.sender.state = send & t.sender.ab = s.sender.ab &
        t.sender.data = s.sender.data &
        t.s2r.in = s.s2r.in & t.r2s.out = s.r2s.out;
      s.sender.state = send :
      . . .
  );
bool Trans(State s, State t) TransSender(s,t) | . . . ;
```

One (weak) property we want to verify is that it is always possible that the control state of the sender will eventually be `get` – or AG EF `sender.state = get` as CTL formula. Translated to the μ-calculus using the optimization of forward analysis (the `-f` option of SMV) this results in the definition of four recursive predicates:

```
mu bool Reachable(State s)
  Start(s) | (exists State t. Trans(t,s) & Reachable(t));
mu bool EF_sender_state_get(State s)
  Reachable(s) & (s.sender.state = get |
  (exists State t. Trans(s,t) & EF_sender_state_get(t)));
nu bool AG_EF_sender_state_get(State s)
  Reachable(s) & (EF_sender_state_get(s) &
  (forall State t. Trans(s,t) -> AG_EF_sender_state_get(t)));
forall State s. Start(s) -> AG_EF_sender_state_get(s);
```

The model checker μcke now evaluates this last line and answers with true or false. If the user wants to have a counter example or a witness for the formula he must request this separately. Other optimizations mentioned in the introduction can be handled the same way as equivalence preserving term rewriting rules.

Performance

We translated our formulation of the alternating bit protocol into the input language of the SMV system and verified the property AG AF `sender.state = get` under fair execution of all four processes and fair channels. The performance under forward analysis of the SMV system and μcke with the same algorithm (μcke) and with simplifying the transition relation with the "restrict" operator of [8] (μcke restrict) is shown in the following table (on a Pentium 120). Also a comparison of the performance of μcke for the scheduler of Milner with [11] on the same machine (Sun 4/75) can be found.

#bits	SMV		μcke		μcke restrict	
	MB	sec	MB	sec	MB	sec
4	9.1	13.3	3.3	9.6	2.9	3.0
5	9.4	36	4.0	42	3.4	7.2
6	10.0	77	5.6	112	4.6	16.2
7	11.3	202	8.8	289	6.0	49.2
8	14.4	696	17	807	12.5	122.3

#	[11]	μcke	μcke restrict
	sec	sec	sec
12	145	21.7	17.2
14	233	31.2	22.6
16	348	39.1	29.4
18	569	54.7	38.1
20	850	67.6	46.5

Conclusion

The μcke model checker shows that a μ-calculus model checker can be as efficient as special purpose model checkers. Currently we investigate how to handle functions with other range types than boolean. Also we look for a way to include unions and inheritance into our type system. See `http://iseran.ira.uka.de/~armin` for more information about μcke or contact the author.

References

1. K. Bartlett, R. Scantlebury, and P. Wilkinson. A note on reliable full-duplex transmissions over half-duplex lines. *Communications of the ACM*, 5(2):260–261, 1969.
2. I. Beer, S. Ben-David, D. Geist, R. Gewirtzman, and M. Yoeli. Methodology and system for practical formal verification of reactive hardware. In Dill [9], pages 182–193.
3. A. Biere. *Efficient μ-Calculus Model Checking with Binary Decision Diagrams*. PhD thesis, Fakultät für Informatik, Universität Karlsruhe, Germany, Jan. 1997. In German. To appear.
4. J. R. Burch, E. M. Clarke, D. E. Long, K. L. McMillan, and D. L. Dill. Symbolic model checking for sequential circuit verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(4):401–424, Apr. 1994.
5. J. R. Burch, E. M. Clarke, and K. L. McMillan. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98:142–170, 1992.
6. 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, April 1986.
7. E. M. Clarke, O. Grumberg, H. Hiraishi, S. Jah, D. E. Long, K. L. McMillan, and L. A. Ness. Verification of the futurebus+ cache coherence protocol. *Formal Methods in System Design*, 6:217–232, 1995.
8. O. Coudert and J. C. Madre. A unified framework for the formal verification of sequential circuits. In *IEEE Intl. Conference on Computer-Aided Design*, pages 126–129, 1990.
9. D. L. Dill, editor. *Computer Aided Verification, 6th International Conference, CAV'94*, volume 818 of *LNCS*. Springer-Verlag, June 1994.
10. Á. T. Eiríksson and K. L. McMillan. Using formal verification/analysis methods on the critical path in system design: A case study. In Wolper [20], pages 367–380.
11. R. Enders, T. Filkorn, and D. Taubner. Generating BDDs for symbolic model checking. *Distributed Computing*, 6:155–164, 1993.
12. G. Janssen. ROBDD software. Technical report, Department of Electrical Engineering, Eindhoven University of Technology, Oct. 1993.
13. A. Kick. *Generation of Counterexamples and Witnesses for Model Checking*. PhD thesis, Fakultät für Informatik, Universität Karlsruhe, Germany, July 1996.
14. K. L. McMillan. *Symbolic Model Checking*. Kluwer, 1993.
15. V. G. Naik and A. P. Sistla. Modeling and verification of a real life protocol using symbolic model checking. In Dill [9], pages 194–206.
16. J. Philipps and P. Scholz. Formal verification of statecharts with instantaneous chain reactions. In *TACAS'97*, 1997.
17. S. Rajan, N. Shankar, and M. K. Srivas. An integration of model checking with automated proof checking. In Wolper [20], pages 84–97.
18. A. Rauzy. Toupie = μ-calculus + constraints. In Wolper [20], pages 114–126.
19. R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *IEEE Intl. Conference on Computer-Aided Design*, pages 42–47, 1993.
20. P. Wolper, editor. *Computer Aided Verification, 7th International Conference, CAV'95*, volume 939 of *LNCS*. Springer-Verlag, July 1995.