Submission to CAV 2000 [Category B: Tool presentation]

XMC: A Logic-Programming-Based Verification Toolset*

C. R. Ramakrishnan, I. V. Ramakrishnan, Scott A. Smolka
with
Yifei Dong, Xiaoqun Du, Abhik Roychoudhury, and V. N. Venkatakrishnan

Department of Computer Science, SUNY at Stony Brook
Stony Brook, NY 11794–4400, USA

Abstract. XMC supports the specification, simulation, and verification of concurrent systems such as communication protocols and embedded systems. It is implemented atop XSB, a high-performance logic-programming system. System models are specified in XL, a typed value-passing language based on Milner's CCS, properties of interest are specified in the modal mu-calculus, and model checking is used to verify properties of systems. XMC incorporates a *justifier* which allows the user to navigate the proof tree underlying a model-checking computation; such proof trees are effective in debugging branching-time formulas. XMC has been successfully applied to the specification and verification of a variety of systems including the Rether real-time Ethernet protocol, the Java meta-locking algorithm, and the SET e-commerce protocol.

Contact author: C.R. Ramakrishnan
Department of Computer Science
SUNY at Stony Brook
Stony Brook, NY 11794-4400, USA
E-mail: cram@cs.sunysb.edu

* Research supported in part by NSF grants EIA-9705998, CCR-9711386, CCR-9805735, and CCR-9876242.

1 Introduction

XMC is a toolset for specifying, simulating, and verifying concurrent systems.[1] Its main mode of verification is temporal-logic model checking [CES86], although equivalence checkers have also been implemented. In its current form, temporal properties are specified in the alternation-free fragment of the modal mu-calculus [Koz83], and system models are specified in XL, a value-passing language based on Milner's CCS [Mil89]. The core computational components of the XMC system, such as those for compiling the specification language, model checking, etc., are built on top of the XSB tabled logic-programming system [XSB98].

A distinguishing aspect of XMC is that model checking is carried out as *query evaluation*, by building proof trees using tabled resolution. The main advantage to making proof-tree construction central to XMC is the resultant flexibility and extensibility of the system. For example, XMC provides the foundation for the XMC-RT [DRS99] model checker for real-time systems, and for XMC-PS [RKR+00], a verification technique for parameterized systems. Secondly, it paves the way for building an effective and uniform interface, called the *justifier*, for debugging branching-time properties, as well as for viewing the results of simulation.

The main features of the XMC system are as follows.

- The specification language, XL, extends value-passing CCS with parameterized processes, first-class channels, logical variables and computations, and supports SML-like polymorphic types.
- XL specifications are compiled into efficient automata representations using techniques described in [DR99]. XMC implements an efficient, local model checker that operates over these automata representations. The optimization techniques in the compiler make the model checker comparable, in terms of performance, to SPIN [HP96] and Murphi [Dil96].
- The model checker is declaratively written in under 200 lines of XSB tabled Prolog code [RRR+97]. XSB's tabled-resolution mechanism automatically yields an on-the-fly, local model checker. Moreover, state representation using Prolog terms yields a form of data-independence [Wol86], permitting model checking of certain infinite-state systems.
- The model checker saves "lemmas", i.e. intermediate steps in the proof of a property. The XMC justifier extracts a proof tree from these lemmas and permits the user to interactively navigate through the proof tree. The same navigation interface is used for simulation.

The XMC system has been successfully used for specifying and verifying different protocols and algorithms such as Rether [CV95], an Ethernet-based protocol supporting real-time traffic; the Java meta locking algorithm [ADG+99,BSW00], a low-overhead mutual exclusion algorithm used by Java threads; and the SET protocol [SET97], an e-commerce protocol developed for Visa/MasterCard.

Below we describe the salient features of the XMC system.

[1] See `http://www.cs.sunysb.edu/~lmc` for details on obtaining a copy of the system.

2 XL: The Specification Language

XL is a language for specifying asynchronous concurrent systems. It inherits the parallel composition (written as '|'), and choice operators ('#'), the notion of channels, input ('!') and output ('?') actions, and synchronization from Milner's value-passing calculus. XL also has a sequential composition (';'), generalizing CCS's prefix operation, and a builtin conditional construct ('if'). XL's support of parameterized processes fills the roles of CCS-style restriction and relabeling.

Complex processes may be defined starting from the elementary actions using these composition operations. Process definitions may be recursive; in fact, as in CCS, recursion is the sole mechanism for defining iterative processes. Processes take zero or more parameters. Process invocations bind these parameters to values: data or channel names.

Data values may be constructed out of primitive types (integers and boolean), predefined types such as lists (written as `[Hd|Tl]` and `[]` for empty list) or arbitrary user-defined (possibly recursive) types. XL provides primitives for manipulating arithmetic values; user-defined computation may be specified directly in XL, or using inlined Prolog predicates. Some of these features are illustrated by the following specification of a FIFO channel having an unbounded buffer:

```
chan(Read, Write, Buf) ::=
        receive(Read, Write, Buf)
     #  {Buf \== []; send(Read, Write, Buf)}.

receive(Read, Write, Buf) ::=
        Read?Msg; chan(Read, Write, [Msg|Buf]).

send(Read, Write, Buf) ::=
        strip_from_end(Buf, Msg, RNBuf);
        Write!Msg;
        chan(Read, Write, RNBuf).

{* %Inlined Prolog code appears between the braces
strip_from_end([X], X, []).
strip_from_end([X,Y|Ys], Z, [X|Zs]) :-
        strip_from_end([Y|Ys], Z, Zs).
*}
```

Type declarations are not always necessary, as the above example illustrates. XMC's type-inference module automatically infers the most general types for the different entities in the specification.

3 The XMC Compiler and Model Checker

In addition to static checkers such as the type checker mentioned above, the XMC system incorporates an optimizing compiler that translates high-level XL specifications into *rules* representing the global transition relation of the underlying automaton. The transitions can be computed from these rules in unit time (modulo indexing) during verification. The compiler incorporates several optimizations to reduce the state space of the generated automaton. One optimization combines computation steps across boundaries of basic blocks, which cannot be done based on user annotations alone, and has been shown as particularly effective [DR99].

The mu-calculus model checker in XMC is encoded using a predicate `models` which verifies whether a state represented by a process term models a given modal mu-calculus formula. This predicate directly encodes the natural semantics of the modal mu-calculus [RRR+97]. The encoding reduces model checking to logic-program query evaluation; the goal-directed evaluation mechanism of XSB ensures that the resultant model checker is local.

Various statistics regarding a model-checking run, such as the memory usage, may be directly obtained using primitives provided by the underlying XSB system. In addition, certain higher-level statistics, such as the total number of states in the system, are provided by the XMC system.

4 Justifier

Tabled resolution of logic programs proceeds by recording subgoals ("lemmas") and their provable instances in tables. Thus, after a goal is resolved, the relevant parts of the proof tree can be reconstructed by inspecting the tables themselves. In XMC, model checking is done by resolving a query to the `models` predicate. The justifier inspects the tables after a model-checking run to create a *justification* tree: a representation of the proof tree or the set of all failed proof paths, depending on whether the verification succeeded or failed, respectively.

The justification tree is usually too large for manual inspection. Hence XMC provides an interactive proof-tree navigator which permits the user to expand or truncate subtrees of the proof. Each node in the proof tree corresponds to computing a single-step transition or a subgoal to the `models` predicate; at each node the justifier interface shows the values of the program counters and other variables of each local process corresponding to the current global state.

Simulation of a system involves following a path through the global transition graph. Such paths can be viewed as a "justification" run for the reachability relation. The XMC system provides a simple simulation interface based on this observation. A system is simulated by first computing the reachability relation and then using the justifier interface to navigate through its proof tree.

5 Future Work

Work to extend the XMC system is proceeding in several directions. First, we are adding a local LTL model checker to the system. Secondly, we are expanding the class of systems that can be verified by incorporating a model checker for real-time systems, XMC-RT [DRS99] built by adding a constraint library to XSB. Thirdly, we plan to include deductive capabilities to XMC by incorporating our recent work in automatically constructing induction proofs for verifying parameterized systems [RKR+00]. Finally, we are enhancing the proof-tree navigator by integrating message sequence charts for better system visualization.

References

[ADG+99] O. Agesen, D. Detlefs, A. Garthwaite, R. Knippel, Y. S. Ramakrishna, and D. White. An efficient meta-lock for implementing ubiquitous synchronization. In *Proceedings of OOPSLA '99*, 1999.

[AH96] R. Alur and T. A. Henzinger, editors. *Computer Aided Verification (CAV '96)*, volume 1102 of *Lecture Notes in Computer Science*, New Brunswick, New Jersey, July 1996. Springer Verlag.

[BSW00] S. Basu, S. A. Smolka, and O. R. Ward. Model checking the Java Meta-Locking algorithm. In *Proceedings of 7th IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2000)*, Edinburgh, Scotland, April 2000.

[CES86] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM TOPLAS*, 8(2), 1986.

[CV95] T. Chiueh and C. Venkatramani. The design, implementation and evaluation of a software-based real-time ethernet protocol. In *Proceedings of ACM SIGCPMM'95*, pages 27–37, 1995.

[Dil96] D. L. Dill. The Murφ verification system. In Alur and Henzinger [AH96], pages 390–393.

[DR99] Y. Dong and C. R. Ramakrishnan. An optimizing compiler for efficient model checking. In *Proceedings of FORTE/PSTV '99*, 1999.

[DRS99] X. Du, C. R. Ramakrishnan, and S. A. Smolka. Tabled resolution + constraints: A recipe for model checking real-time systems. Technical report, Department of Computer Science, SUNY, Stony Brook, 1999. URL: `http://www.cs.sunysb.edu/~cram/papers/manuscripts/rtlmc`.

[HP96] G. J. Holzmann and D. Peled. The state of SPIN. In Alur and Henzinger [AH96], pages 385–389.

[Koz83] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[Mil89] R. Milner. *Communication and Concurrency*. International Series in Computer Science. Prentice Hall, 1989.

[RKR+00] A. Roychoudhury, K. Narayan Kumar, C.R. Ramakrishnan, I.V. Ramakrishnan, and S.A. Smolka. Verification of parameterized systems using logic-program transformations. In *Proc. of TACAS 2000*. Springer-Verlag, 2000.

[RRR+97] Y. S. Ramakrishna, C. R. Ramakrishnan, I. V. Ramakrishnan, S. A. Smolka, T. W. Swift, and D. S. Warren. Efficient model checking using tabled resolution. In *Proceedings of the 9th International Conference on Computer-Aided Verification (CAV '97)*, Haifa, Israel, July 1997. Springer-Verlag.

[SET97] SET Secure Electronic Transaction LLC. *The SET Standard Specification*, May 1997. URL: `http://www.setco.org/set_specifications.html`.

[Wol86] P. Wolper. Expressing interesting properties of programs in propositional temporal logic. In *Proc. 13th ACM Symp. on Principles of Programming Languages*, pages 184–192, St. Petersburgh, January 1986.

[XSB98] The XSB Group. The XSB logic programming system v2.01, 1998. Available from `http://www.cs.sunysb.edu/~sbprolog`.