Maria: Modular Reachability Analyser for Algebraic System Nets

Marko Mäkelä[*]

Helsinki University of Technology, Laboratory for Theoretical Computer Science
P.O.Box 9700, 02015 HUT, Finland
marko.makela@hut.fi
http://www.tcs.hut.fi/Personnel/marko.html

Abstract. Maria performs simulation, exhaustive reachability analysis and on-the-fly LTL model checking of high-level Petri nets with fairness constraints. The algebra contains powerful built-in data types and operations. Models can be exported to low-level Petri nets and labelled transition systems. Translator programs allow Maria to analyse transition systems as well as distributed computer programs written in procedural or object-oriented languages, or high-level specifications such as SDL. Maria has been implemented in portable C and C++, and it is freely available under the conditions of the GNU General Public License.

1 Introduction

1.1 Analysing High-Level Software Systems

There are many tools for analysing concurrent systems, but most of them are only suitable for education or for analysing relatively simple, hand-made highly abstracted models. At universities, many analysers have been developed just to see whether a theoretical idea might work in practice, often analysing models that do not directly have any roots in the real world. Commercial tool vendors concentrate on executable code generation and on graphical user interfaces.

Verifying industrial-size designs with minimal manual effort is a challenge. There may be no universal solution, but it is possible to list some requirements for automated checking of distributed software systems.

High-level formalism. The formalism used by the reachability analyser or model checker should have enough expressive power, so that high-level system descriptions can be modelled in a straightforward way, without introducing any superfluous intermediate states caused by having to translate, e.g., message buffer operations to non-atomic sequences of simpler operations.

[*] This research was financed by the Helsinki Graduate School on Computer Science and Engineering, the National Technology Agency of Finland (TEKES), the Nokia Corporation, Elisa Communications, the Finnish Rail Administration, EKE Electronics and Genera, and by a personal grant from Tekniikan Edistämissäätiö.

J. Esparza and C. Lakos (Eds.): ICATPN 2002, LNCS 2360, pp. 434–444, 2002.

Ease of use. Users should not need to be familiar with the formalism internally used by the analyser. The user works in the domain he is used to, and a *language-specific front-end* is responsible for hiding the underlying formalism:

– translate models to the internal formalism, abstracting from details that are not necessary in analysis
– allow desired properties to be specified in the design domain
– display erroneous behaviour in the design domain

Efficient utilisation of computing resources. The tools used in the modelling and verification process should be constructed in such a way that they work in a variety of computer systems, ranging from personal computers to multiprocessor supercomputers. The tools should not depend on the processor word length or byte order, and they should be based on standardised interfaces, such as [28]. Memory management should be optimised for large numbers of states and events.

1.2 Representing State Spaces

Efficient algorithms for exhaustive state space enumeration need to determine whether a state has been encountered earlier in the analysis. There are three fundamentally different approaches for representing the set of covered states.

Symbolic Techniques represent the set with a dynamic data structure that may be changed substantially when a state is added.
Explicit Techniques encode each state separately as a bit string.
Lossy Techniques transform states to hash signatures. In the event of a hash collision, substantial parts of the state space may remain unexplored.

It may be difficult to combine symbolic techniques with efficient model checking of liveness properties. Also, symbolic techniques have usually been implemented for formalisms having relatively simple data types and operations.

Since Maria supports a very high-level formalism, including, among others, operations on bounded queues, no efforts were made to implement symbolic techniques. When the state space management is based on explicit or lossy techniques, new operations on existing data types can be implemented independently.

1.3 Background

Maria [20] is the main product of a four-year research project that was carried out during the years 1998–2001. One of the most important goals in the project is to be able to directly analyse telecommunications protocols specified in SDL [29], the CCITT Specification and Description Language. In fact, the experience from Prod [26] and Emma [8] motivated the start of the whole project.

The goal was to develop a Petri net based state space exploration tool whose inscription language facilitates straightforward translation of data types and constructs found in SDL and high-level programming languages. Despite its expressive power, the Maria modelling language has a sound theoretical foundation. The semantics has been defined in [17] in terms of Algebraic System Nets [10].

The first usable versions of the analyser were released in the summer of 1999. Since then, MARIA has been in extensive internal use, which has helped in finding and correcting errors and performance bottlenecks. In 2001, a graphical user interface for exploring state spaces and displaying query results was implemented on top of GraphViz [3]. In the fall of 2001, MARIA replaced PROD as the main analysis tool in the education at our laboratory. On November 1, 2001, the tool was officially released as version 1.0.

The word "modular" in MARIA refers to the software design of the tool, which makes it easy to incorporate different algorithms, front-ends and state storage mechanisms. The work on refinements [14] demonstrates that MARIA can be used as a test-bed for new algorithmic ideas.

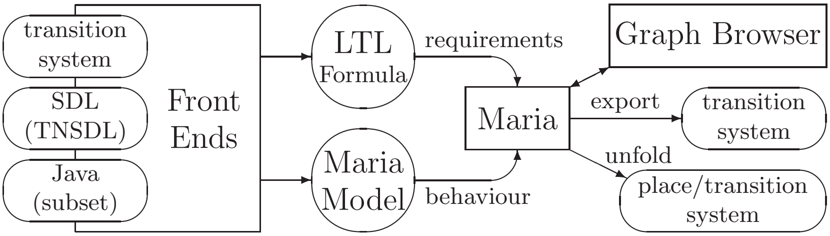

Fig. 1. The interfaces of MARIA

2 Using Maria

Figure 1 illustrates the high-level interfaces of MARIA. Models can be either written by hand or translated from other formalisms. A translator that allows MARIA to model check parallel compositions of TVT [25] labelled transition systems is available from the home page [20]. Translators from SDL and Java are under development. The SDL translator [23] has been written from scratch, while one Java translator is based on Bandera [2].

MARIA accepts commands from files, from a command line and from a graphical interface. Several modes of operation are supported:

- exhaustive reachability analysis with on-the-fly checking of safety properties
- interactive simulation: generate successors for states selected by the user
- interactive reachability graph exploration
- on-the-fly verification of liveness properties with fairness assumptions
- unfolding the model, optionally using a "coverable marking" algorithm [18]

The unfolding algorithms can output nets in the native input formats of PEP [7] and LoLA [22], as well as in the native format of PROD.

Figure 2 shows a simplified version of the distributed data base manager system, originally presented by Genrich and Lautenbach in [5]. The number of

```
typedef id[3] db_t;              trans receive
typedef struct {                 in  { place inactive: r;
  db_t first;                          place sent: { s, r }; }
  db_t second;                   out { place performing: r;
} db_pair_t;                           place recv: { s, r }; };

place waiting db_t;              trans ack
place performing db_t;           in  { place performing: r;
place inactive db_t: db_t d: d;        place recv: { s, r }; }
place exclusion struct {}: {};   out { place inactive: r;
place sent db_pair_t;                  place ack: { s, r }; };
place recv db_pair_t;
place ack db_pair_t;

trans collect
in  { place waiting: s;
      place ack: db_t t (t != s): { s, t }; }
out { place inactive: s; place exclusion: {}; };

trans update
in  { place inactive: s; place exclusion: {}; }
out { place waiting: s;
      place sent: db_t t (t != s): { s, t }; };
```

Fig. 2. Distributed data base managers in the Maria language. The number of data base manager nodes can be configured by modifying the first line

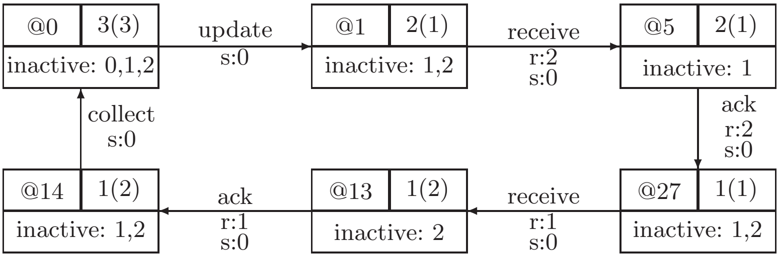

Fig. 3. An error trace for the LTL formula `<>place inactive equals empty` in the system of Figure 2. In this infinite execution, always at least one node is inactive

modelled nodes can be changed by altering the first data type definition; Maria allows aggregate inscriptions in the transitions `collect` and `update`.

One use of Maria is to verify liveness properties, such as the (incorrect) claim that in all executions starting from the initial state, all data base nodes are simultaneously active at some point of time. Error traces can be simplified by telling Maria to only show the markings of certain places. The error trace in Figure 3 hides everything except the place named `inactive`.

3 Advanced Features

3.1 Powerful Algebraic Operations

The built-in algebraic operations in Maria were designed to have enough expressive power for modelling high-level programs. In addition to the basic constructs familiar from programming languages such as C, there are operations for:

- managing items in bounded buffers (queues and stacks)
- basic multi-set operations (union, intersection, difference, mappings)
- aggregation: multi-set sums, existential and universal quantification

Aggregation over a dynamic range of indexes is a particularly powerful construct, because it allows arc expressions to be highly parameterisable. In Figure 2, the transition `update` models a broadcast operation with multi-set summation. For instance, when the variable `s` equals `3`, the formula `db_t t (t!=s): {s,t}` expands to the multi-set `{s,1},{s,2}`, or `{3,1},{3,2}`. If the first line of Figure 2 is modified to model n data base nodes, the multi-set will have $n-1$ elements.

Basic algebraic operations check for exceptional conditions, such as integer arithmetic errors, queue or stack overflows or underflows, and constraint violations. The data type system is very sophisticated, and it is possible to arbitrarily restrict the set of allowed values even for structured types.

3.2 Optional C Code Generation

Transforming Maria models to executable libraries is a nontrivial task, because operations on multi-sets, queues, stacks and tagged unions have no direct counterparts in the C programming language. In addition, all code must be instrumented with guards against evaluation errors and constraint violations.

The use of the compilation option [16] can speed up all operations except unfolding. Interpreter-based operation is useful in interactive simulations or when debugging a model, because the overhead of invoking a compiler is avoided.

3.3 Unifying Transition Instances and Markings

The unification algorithm that Maria uses for determining the assignments under which transitions are enabled has been documented in [18]. In Maria, this depth-first algorithm has been implemented in such a way that enabled transition instances are fired as soon as they are found.

The interpreter-based implementations of the combined transition enabling check and firing algorithm use a search stack, while the compiler-based variant integrates the search stack in the program structure as nested loops.

3.4 Efficient State Space Management

Maria represents the states of its models in two different ways. The expanded representation is used when determining successor states and performing computations. Long-term storage of explored states is based on a condensed representation, a compact bit string whose encoding has been documented in [15].

By default, Maria manages reachability graphs (reachable states and events) in disk files. Keeping all data structures on disk has some advantages:

- the analysis can be interrupted and continued later
- the generated reachability graph can be explored on a different computer
- memory capacity is not a limit: a high-level model with 15,866,988 states and 61,156,129 events was analysed in 5 MB of RAM (and 1.55 GB of disk)

File system access can be notably slow even on systems that have enough memory to buffer all files. In some cases, more than half of the execution time of the analyser can be spent in determining whether a state has been visited. Enabling an option for memory-mapped file access reduces the analysis times of certain models to a sixth of the original. Unfortunately, with this option, the 4 GB address space of 32-bit systems is becoming a barrier. Really complex models can be analysed only with traditional file access, or with a 64-bit processor.

Optionally, Maria approximates the set of reachable states with a memory-based hash table. This option can be useful in cursory analysis of deadlocks and safety properties and for obtaining lower bounds for state space sizes.

3.5 Model Checking with Fairness Constraints

The on-the-fly LTL model checker in Maria takes into account both weak and strong fairness on the algorithmic level instead of adding it to the LTL specification. When a model has many fairness constraints, this can lead to exponential savings in time and space. To this end, the net class of the analyser includes constructs to flexibly express fairness constraints on transitions. To our knowledge, this is the first model checker of its kind for high-level Petri nets.

The model checking procedure [12,13] proceeds in an on-the-fly manner processing one strongly connected component of the product state space at a time. By detecting the presence of strong fairness constraints, the procedure tries to use generalised Büchi automata when possible to avoid the more complex task of checking the emptiness of Streett automata.

Generating short counterexamples when both weak and strong fairness are present can be quite challenging. Experiments with the algorithm that Maria uses have been reported in [12].

For translating temporal properties to generalised Büchi automata Maria invokes an external tool that is a highly optimised implementation of the algorithm presented in [4]. Since the translator works as a textual filter, it can be easily replaced with an implementation of another algorithm.

4 Performance and Applications

4.1 Distributed Dynamic Channel Allocation

One of the first systems that was analysed with Maria is a model of a radio channel allocation algorithm [21].

The sophisticated data type system and the aggregation operations of the modelling language allowed to write the model in a very compact way, and many modelling errors were quickly found in interactive simulation runs.

4.2 Verbatim Modelling of a Large SDL Specification

An important goal of Maria was to be able to handle models of industrial-size distributed systems. The most complex system that has been analysed with Maria so far is the *complete* radio link control (RLC) protocol of the third-generation mobile telephone system UMTS. The ETSI standard [27] describes this protocol in English, and the description is accompanied with 74 pages of informative graphical SDL diagrams that contain some inaccuracies and errors.

Each SDL statement was mechanically translated into a high-level net transition, generating hundreds of transitions. As one of the main functions of the protocol is to disassemble and reassemble data packets, the transmitted messages had to be modelled in detail. For one version of the model, Maria encodes each reachable marking of the 142 high-level places in 167–197 bytes.

Timers were modelled in an abstract way, because the formalism of Maria cannot describe time, but only the order in which events occur. Each timer was translated to a Boolean flag that indicates whether the timer is active. When a timer is activated, the flag is set. Resetting the timer clears the flag. Whenever the flag is set, a timeout can occur.

The parameters of the protocol model include message queue lengths, the domains of sequence numbers, and the type of communication channels. Both a reliable and a lossy channel have been analysed, but so far, the protocol has not been analysed on a channel that would duplicate or reorder messages. With the analysed initial parameters, the model has up to tens of millions of reachable states. LTL model checking has been applied on configurations that have less than 100,000 reachable states. The results will be reported in [24].

4.3 Benchmark: Distributed Data Base Managers

Although Maria has a higher-level modelling language than Prod, using it does not imply a noticeable performance penalty. In the contrary, Maria usually uses less memory (or disk) than Prod, and sometimes it consumes less processor time.

We have translated the Prod model of 10 distributed data base servers (sample file `dbm.net`) to Maria format (file `dbm.pn` in the Maria distribution). Figure 2 is a simplification of this model; among other things, it excludes capacity constraints and invariants that allow more compact representation of markings.

When Prod was invoked with a reasonably large `-b` parameter, it generated the 196,831 states and 1,181,000 arcs of the 10-server model in 262 seconds of user time and 10 seconds of system time. The 700 MHz Pentium III system had enough available memory to buffer the 69 megabytes of generated graph files.

We analysed the equivalent model in Maria with the compilation option enabled. The analysis produced 22 megabytes of graph files in 130 seconds of user time and less than 1 second of system time. The improvement can be attributed to better state space encoding and to utilising a memory-mapped interface [28, Section 2.8.3.2] for accessing graph files.

5 Availability

Maria is freely available as source code from the home page [20] under the conditions of the GNU General Public License. No registration is required.

It should be possible to compile the analyser for any environment that fully supports the C and C++ programming languages. Some features are, however, only present on Unix-like systems. The tool has been tested on GNU/Linux, Sun Solaris, Digital UNIX, SGI IRIX, Apple Mac OS X and Microsoft Windows.

6 Conclusion and Future Work

Maria helps the analysis of industrial-size systems in multiple ways:

- by providing a theoretically sound modelling language that is suitable for describing high-level software systems,
- as an interactive simulator and visualiser of distributed systems, and
- as a “model checking back-end” for various formalisms ranging from labelled transition systems to SDL

Currently, Maria does not support any state space reduction methods. Some algorithms on symmetry reduction [9] are being implemented. It has been planned to adapt partial order reduction methods [6] to Maria.

The model checker in Maria could treat safety properties in a more efficient way. Instead of interpreting the reachability graph, the property and their product as Büchi or Streett automata, it could interpret them as finite automata and thus avoid the costly loop checks. A translation from LTL to finite automata has been given in [11]. Recent developments in this area are documented in [19].

The present version of Maria displays counterexamples as directed graphs whose nodes are reachable states in the model and edges are transition instances leading from a state to another. In many applications, it would be more intuitive to illustrate executions with message sequence charts [30]. In order for this to work nicely, a mechanism for specifying mappings from transition instances to messages and from markings to MSC conditions must be implemented.

Acknowledgements

The LTL model checker algorithm in Maria was originally implemented by Timo Latvala. The design of Maria has been influenced by feedback from Nisse Husberg, Teemu Tynjälä, Kimmo Varpaaniemi and others. The author would also like to express his thanks to the anonymous referees for their comments.

References

1. José-Manuel Colom and Maciej Koutny, editors, *Application and Theory of Petri Nets 2001, 22nd International Conference, ICATPN 2001*, volume 2075 of *Lecture Notes in Computer Science*, Newcastle upon Tyne, England, June 2001. Springer-Verlag. 443
2. James. C. Corbett, Matthew. B. Dwyer, John Hatcliff, Shawn Laubach, Corina S. Păsăreanu, Robby, and Hongjun Zheng. Bandera: Extracting finite-state models from Java source code. In Carlo Ghezzi, Mehdi Jazayeri and Alexander Wolf, editors, *Proceedings of the 22nd International Conference on Software Engineering*, pages 439–448, Limerick, Ireland, June 2000. ACM Press, New York, NY, USA. 436
3. Emden R. Gansner and Stephen C. North. An open graph visualization system and its applications to software engineering. *Software: Practice and Experience*, 30(11):1203–1233, September 2000. 436
4. Rob Gerth, Doron Peled, Moshe Y. Vardi, and Pierre Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Proceedings of the 15th Workshop Protocol Specification, Testing, and Verification*, Warsaw, June 1995. North-Holland. 440
5. Hartmann J. Genrich and Kurt Lautenbach. The analysis of distributed systems by means of Predicate/Transition-Nets. In Gilles Kahn, editor, *Semantics of Concurrent Computation*, volume 70 of *Lecture Notes in Computer Science*, pages 123–146, Evian, France, July 1979. Springer-Verlag, 1979. 436
6. Patrice Godefroid, Doron Peled and Mark Staskauskas. Using partial-order methods in the formal validation of industrial concurrent programs. *IEEE Transactions on Software Engineering*, 22(7):496–507, July 1996. 441
7. Bernd Grahlmann. The state of PEP. In Armando M. Haeberer, editor, *Algebraic Methodology and Software Technology, 7th International Conference, AMAST'98, Amazonia, Brazil*, volume 1548 of *Lecture Notes in Computer Science*, pages 522–526, Manaus, Brazil, January 1999. Springer-Verlag. 436
8. Nisse Husberg and Tapio Manner. Emma: Developing an industrial reachability analyser for SDL. In *World Congress on Formal Methods*, volume 1708 of *Lecture Notes in Computer Science*, pages 642–661, Toulouse, France, September 1999. Springer-Verlag. 435
9. Tommi Junttila. Finding symmetries of algebraic system nets. *Fundamenta Informaticae*, 37(3):269–289, February 1999. 441
10. Ekkart Kindler and Hagen Völzer. Flexibility in algebraic nets. In Jörg Desel and Manuel Silva, editors, *Application and Theory of Petri Nets 1998: 19th International Conference, ICATPN'98*, volume 1420 of *Lecture Notes in Computer Science*, pages 345–364, Lisbon, Portugal, June 1998. Springer-Verlag. 435

11. Orna Kupferman and Moshe Y. Vardi. Model checking of safety properties. In Nicolas Halbwachs and Doron Peled, editors, *Computer Aided Verification: 11th International Conference, CAV'99*, volume 1633 of *Lecture Notes in Computer Science*, pages 172–183, Trento, Italy, July 1999. Springer-Verlag. 441
12. Timo Latvala and Keijo Heljanko. Coping with strong fairness. *Fundamenta Informaticae*, 43(1–4):175–193, 2000. 439
13. Timo Latvala. Model checking LTL properties of high-level Petri nets with fairness constraints. In [1], pages 242–262. 439
14. Glenn Lewis and Charles Lakos. Incremental state space construction for coloured Petri nets. In [1], pages 263–282. 436
15. Marko Mäkelä. Condensed storage of multi-set sequences. In *Workshop on the Practical Use of High-Level Petri Nets*, Århus, Denmark, June 2000. 439
16. Marko Mäkelä. Applying compiler techniques to reachability analysis of high-level models. In Hans-Dieter Burkhard, Ludwik Czaja, Andrzej Skowron and Mario Lenz, editors, *Workshop Concurrency, Specification & Programming 2000*, Informatik-Bericht 140, pages 129–141. Humboldt-Universität zu Berlin, Germany, October 2000. 438
17. Marko Mäkelä. A reachability analyser for algebraic system nets. Research report A69, Helsinki University of Technology, Laboratory for Theoretical Computer Science, Espoo, Finland, June 2001. 435
18. Marko Mäkelä. Optimising enabling tests and unfoldings of algebraic system nets. In [1], pages 283–302. 436, 438
19. Marko Mäkelä. Efficiently verifying safety properties with idle office computers. Unpublished manuscript. 441
20. Marko Mäkelä. *Maria*. On-line documentation, `http://www.tcs.hut.fi/maria/`. 435, 436, 441
21. Leo Ojala, Nisse Husberg and Teemu Tynjälä. Modelling and analysing a distributed dynamic channel allocation algorithm for mobile computing using high-level net methods. *International Journal on Software Tools for Technology Transfer*, 3(4):382–393, 2001. 440
22. Karsten Schmidt. LoLA: A low level analyser. In Mogens Nielsen and Dan Simpson, editors, *Application and Theory of Petri Nets 2001, 21st International Conference, ICATPN 2000*, volume 1825 of *Lecture Notes in Computer Science*, pages 465–474, Århus, Denmark, June 2000. Springer-Verlag. 436
23. André Schulz and Teemu Tynjälä. Translation rules from standard SDL to Maria input language. In Nisse Husberg, Tomi Janhunen and Ilkka Niemelä, editors, *Leksa Notes in Computer Science: Festschrift in Honour of Professor Leo Ojala*, Research Report 63, Helsinki University of Technology, Laboratory for Theoretical Computer Science, Espoo, Finland, October 2000. 436
24. Teemu Tynjälä, Sari Leppänen and Vesa Luukkala. Verifying reliable data transmission over UMTS radio interface with high level Petri nets. Unpublished manuscript. 440
25. Antti Valmari et al. Tampere Verification Tool. `http://www.cs.tut.fi/ohj/VARG/`. 436
26. Kimmo Varpaaniemi, Jaakko Halme, Kari Hiekkanen and Tino Pyssysalo. PROD reference manual. Technical Report B13, Helsinki University of Technology, Digital Systems Laboratory, Espoo, Finland, August 1995. 435
27. *Universal Mobile Telecommunications System (UMTS); RLC protocol specification (3GPP TS 25.322 version 3.5.0 Release 1999).* ETSI TS 125 322 V3.5.0 (2000-12). European Telecommunications Standards Institute, December 2000. 440

28. *Standard for Information Technology—Portable Operating System Interface*. IEEE Std 1003.1-2001. Institute of Electrical and Electronics Engineers, New York, NY, USA, December 2001. 435, 441
29. *CCITT Specification and Description Language (SDL)*. Recommendation Z.100. International Telecommunication Union, Geneva, Switzerland, October 1996. 435
30. *Message Sequence Chart (MSC)*. Recommendation Z.120. International Telecommunication Union, Geneva, Switzerland, November 1999. 441