MARIA: Modular Reachability Analyser for Algebraic System Nets

Marko Mäkelä*

May 9, 2001

Abstract

Maria is a tool for analysing concurrent systems. It performs simulation, exhaustive reachability analysis and on-the-fly LTL model checking with fairness constraints. It supports high-level Petri Nets based on an algebra with powerful built-in data types and operations. Optimised for both memory usage and speed, the tool is suitable for analysing models generated from distributed computer programs written in procedural or object-oriented languages, or high-level specifications.

Maria has been implemented in portable C and C++, and it is freely available under the conditions of the GNU General Public License.

Keywords. reachability analysis, high-level nets, concurrent systems, on-the-fly verification

1 Background

There are many tools for analysing concurrent software systems, but most of them are only suitable for education or for analysing relatively simple, hand-made highly abstracted models. Many academic tools 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 user interfaces and code generation. For both camps, exhaustive formal analysis of industrial-size designs using minimal manual effort has been out of the scope for a long time.

The intention behind MARIA [4] is to develop a reachability analyser and model checker for a formalism whose expressive power is close to high-level programming and specification languages (such as C++ and SDL). The powerful queue and stack operations of the language make it possible to model complex communications without introducing superfluous intermediate states that make analysis more difficult.

Users do not need to be familiar with the formalism internally used by the analyser, in our case Algebraic System Nets [1]. The user works in the domain he is used to, and a *language-specific front-end* is responsible for hiding the underlying formalism from the user:

*Laboratory for Theoretical Computer Science, Helsinki University of Technology, P.O. Box 9700, 02015 HUT, Finland, ⟨URL:http://www.tcs.hut.fi/maria/⟩.

- translate user programs to the internal formalism
- allow desired properties to be specified in terms of the application
- display erroneous behaviour as an execution sequence of the application

A generic formalism has some advantages over specialised formalisms. Implementing new analysis methods immediately benefits all languages for which a translation exists.

At the time of writing, we have an experimental front-end for SDL [7]. The data type system of MARIA makes it easy to translate expressions and message passing.

2 Overview

Figure 1 illustrates the modular structure of our analyser. The tool can be run both interactively (textually and graphically) and in batch mode. There are several modes of operation:

- exhaustive reachability analysis
- interactive simulation: generate successors for states selected by the user
- on-the-fly verification of safety and liveness properties specified in temporal logic
- unfolding nets, optionally reduced with a "coverable marking" algorithm [5]

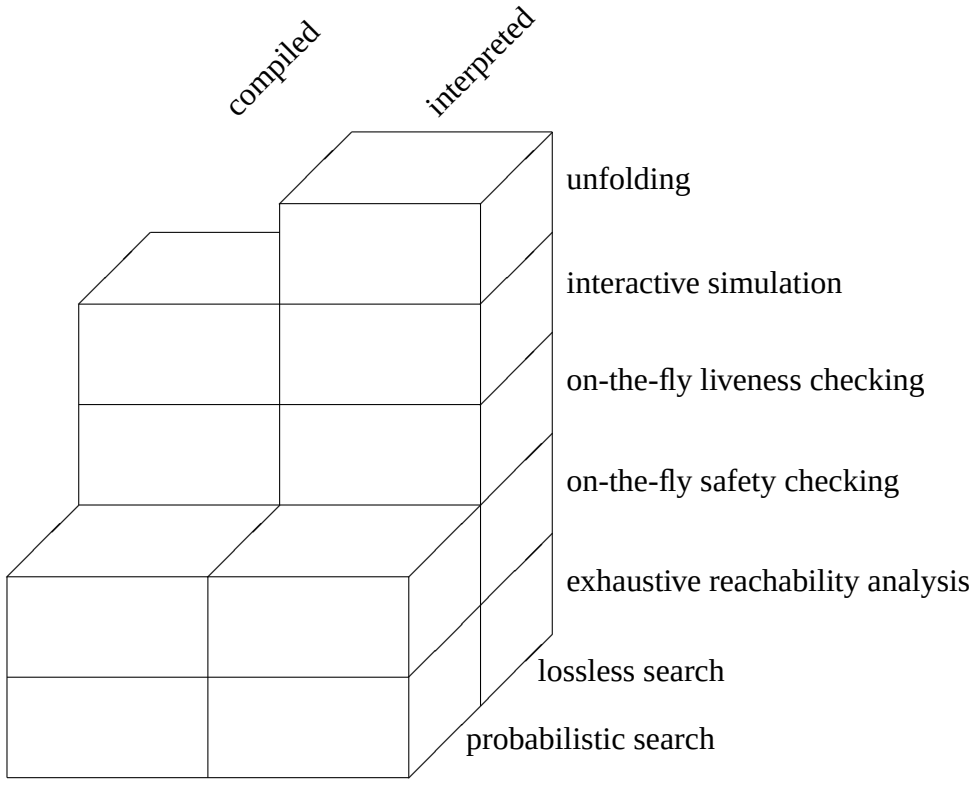

Figure 1: The Features of MARIA

The tool manages the reachability graph (reachable states and actions between states) 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)

The above mentioned analysis was completed in less than 10 hours on a 266 MHz Pentium II system when using the option that generates C code for all model-dependent tasks. Using this option typically reduces analysis times to a third, sometimes to a fifth. The interpreter mode is useful in interactive simulations, when debugging a model.

There is also an experimental implementation of probabilistic verification that does not use disk files. Since the method does not produce any reachability graph, it is most useful when verifying safety properties non-interactively.

3 Graph Exploration

MARIA contains an interactive tool for exploring the reachability graph, for checking temporal logic properties and for performing partial reachability analysis, also called *simulation*.

The reachability graph can be explored and browsed in many ways. Expressions and temporal logic formulae can be evaluated in different states. It is possible to find the shortest path between a specified state and a set of states. The tool can compute the strongly connected components of the graph. Figure 2 illustrates the component graph of a simple model.

Evaluating temporal logic expressions in a partially generated reachability graph will cause new states to be added to the graph, until a counterexample violating the property being verified is found or the property is found to hold.

Simulation can be a useful tool when developing new Petri Net models. The graph explorer performs simulation in a very transparent way: when one lists the successors of an unprocessed state, the successors will be generated, stored in the reachability graph and shown to the user.

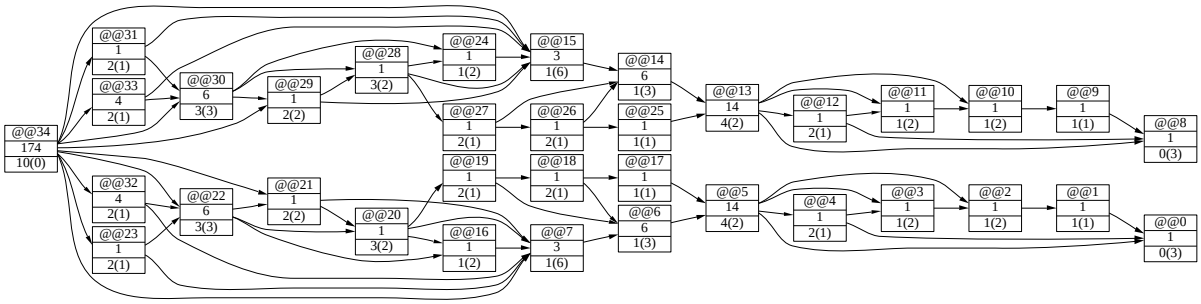

Figure 2: The strongly connected components of a state space

4 Rich Data Types

The data type system was designed with high-level specification and programming languages in mind. Compound types are tuples, arrays indexed by any type, and bounded-capacity queues and stacks. They can be nested and constrained arbitrarily.

For instance, a stack holding 0 or 2 small integers can be defined as `int (1..3) [stack 2] ({},{1,1}..)`. The contents requires $\lceil \log_2(3^0 + 3^2) \rceil = 4$ bits of storage. Tight constraints

- reduce the space occupied for storing states,
- reduce possibilities in exhaustive analysis and when unfolding, and
- helps detect errors in the model (constraint violations).

Detecting constraint violations is useful when analysing manually constructed models. All errors that occur during the analysis are reported.

5 Powerful Algebraic Operations

In addition to basic programming language constructs, there are built-in operations for

- managing buffers (queues and stacks),
- basic multi-set arithmetics (union, intersection, difference, mappings), and
- aggregation: multi-set summation, existential and universal quantification.

Aggregation over a dynamic range of indexes is a particularly powerful construct and extremely useful when modelling distributed algorithms. For instance, it is possible to use compact notation for modelling a server that sends a message to all other servers. In order to change the number of servers in the system, only one data type definition needs to be modified.

Basic algebraic operations check for exceptional conditions, such as overflows or division by zero. All operations detect constraint violations. The set of allowed values can be restricted arbitrarily even for structured types.

6 Model Checking with Fairness Constraints

A fully automated analysis can detect rather simple and coarse errors, such as

- states where the system cannot proceed (deadlocks), and
- erroneous computing steps where some kind of an error occurs.

Experts who know the system can describe the desired properties of the system e.g. with formulae of temporal or modal logic. These properties are usually divided into safety and liveness properties. A violation of a safety property (“nothing bad happens”) is a chain of events from the initial state of the system to a “bad” state that is reachable in the system. A liveness property (“something good eventually happens”) is violated if the system can infinitely execute a loop of events without performing a “good” event.

Liveness properties are often refined with fairness assumptions. For instance, a communication protocol should eventually deliver the sent messages to the recipient. If the communication path may lose or distort messages, we are not interested in such counterexamples where the connection is constantly broken. A fairness assumption can dictate that the events modelling the lossy data path must treat the alternative events fairly.

The on-the-fly LTL model checker in MARIA takes into account both weak and strong fairness constraints specified in the model. Omitting the fairness properties from the LTL formula leads to exponential savings [3]. To our knowledge, this is the first model checker of its kind for high-level Petri nets.

Counterexamples, or event loops that violate liveness properties, easily become even thousands of events long when fairness assumptions are present. MARIA tries to locate short loops efficiently [2].

For translating LTL formulae to generalised Büchi automata we use an external tool [6]. As MARIA is modular, replacing the translator is easy.

References

[1] 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, Berlin, Germany. ⟨URL:http://www.informatik.hu-berlin.de/%7Evoelzer/Flex.html⟩

[2] Timo Latvala and Keijo Heljanko. Coping with strong fairness. *Fundamenta Informaticae*, 43(1–4):175–193, 2000. ⟨URL:http://www.tcs.hut.fi/%7Etimo/FI2000-LatHel.ps.gz⟩

[3] Timo Latvala. Model checking LTL properties of high-level Petri nets with fairness constraints. To appear in *Application and Theory of Petri Nets 2001: 22nd International Conference, ICATPN'01*, Newcastle upon Tyne, England, June 2001.

[4] Marko Mäkelä. Modular reachability analyzer for high-level Petri nets. In René Boel and Geert Stremersch, editors, *Discrete Event Systems: Analysis and Control*. Proceedings of the 5th Workshop on Discrete Event Systems, Ghent, Belgium, August 2000. Kluwer Academic Publishers.

[5] Marko Mäkelä. Optimising enabling tests and unfoldings of algebraic system nets. To appear in *Application and Theory of Petri Nets 2001: 22nd International Conference, ICATPN'01*, Newcastle upon Tyne, England, June 2001.

[6] Mauno Rönkkö. A distributed object oriented implementation of an algorithm converting a LTL formula to a generalised Büchi automaton. On-line documentation, 1999. ⟨URL:http://www.abo.fi/%7Emronkko/PROJECT/LTL2BUCHI/abstract.html⟩.

[7] *CCITT Specification and Description Language (SDL)*. Recommendation Z.100. International Telecommunication Union, Geneva, Switzerland, October 1996.