MARIA: Modular Reachability Analyser for Algebraic System Nets

Marko Mäkelä
Laboratory for Theoretical Computer Science
Helsinki University of Technology
P.O.Box 9700
02015 HUT
Finland

October 29, 2001

Analysis Tools for Concurrent Systems

Systems where concurrency is present may contain sporadic errors that are very difficult to reproduce. This kind of errors may become expensive in distributed systems where large numbers of terminals have been sold to customers.

Not all concurrency-related errors can be found with a debugger or by testing. The fact that tests do not reveal any erroneous behaviour does not prove a system correct. Only if the analysis covers all possible states and executions of the system, we can be sure that the system (or the model that describes it hopefully correctly) fulfils the desired properties (which have hopefully been chosen in a meaningful way).

One method that can be automated easily is *reachability analysis*, exploring all executions or states.

Reachability Analysers at the Helsinki University of Technology

1984–1988: PRENA (thousands of reachable states); Pr/T nets with Pascal inscriptions

1989–: PROD (millions of reachable states); Pr/T nets with C inscriptions

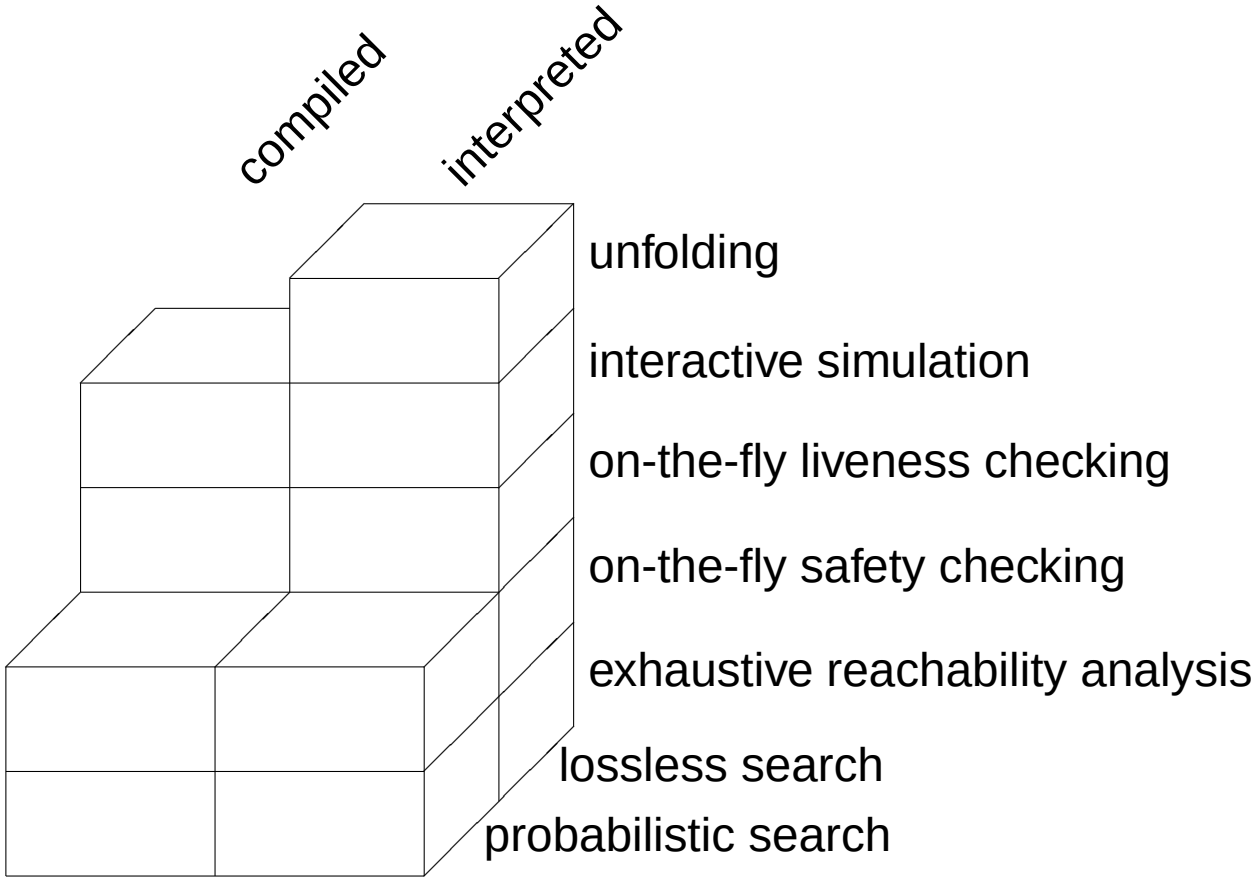
- on-the-fly verification, stubborn sets, . . .

1998–: MARIA (tens of millions of reachable states); algebraic system nets

- two modes of operation: interpreted (C++) and compiled (C/C++)
- modular structure: easy to implement new algorithms

The tools have been used e.g. for analysing two PLC based railway systems, the back-bone bus of an ATM switch, the ISDN DSS.1 protocol and the UMTS RLC protocol.

The Features of MARIA



Different Modes of Operation

MARIA can process model systems in two fundamentally different ways. It can either interpret the models, or it can translate a model into a set of C language modules that are compiled into a machine executable library.

The interpreter approach is useful when a model is being edited and simulated. The calculation of successor states is slower than when the model is compiled, but the delay caused by compiling the model is avoided.

It is best to use the compiler mode only in connection with batch-mode state space exploration.

Graph Algorithms in MARIA

MARIA offers some basic tools for traversing reachability graphs:

- listing the successor and predecessor states of a state,
- computing shortest paths between two states, or a state and a set of states,
- traversing a graph of strongly connected components, and
- illustrating execution paths that violate a desired property.

The paths or graphs can be displayed either textually or graphically.

Interfaces to Other Tools (1/3)

MARIA has a number of interfaces to other tools. Some tools are invoked as subprograms:

- a C compiler and linker for translating models,
- a translator from linear temporal logic (LTL) to generalised Büchi automata, and
- GraphViz by AT&T Labs for illustrating state spaces.

Interfaces to Other Tools (2/3)

Our research group has developed some tools that translate MARIA model systems from other specification languages:

- CCITT Specification and Description Language translator (under development)
- TeleNokia SDL (TNSDL) translator (based on EMMA, a translator from TNSDL to PROD)
- Tampere Verification Tool (TVT) labelled state transition system (LSTS) translator

In addition, a group at Brandenburg University of Technology at Cottbus, Germany, uses MARIA for analysing programmable logic controller (PLC) systems.

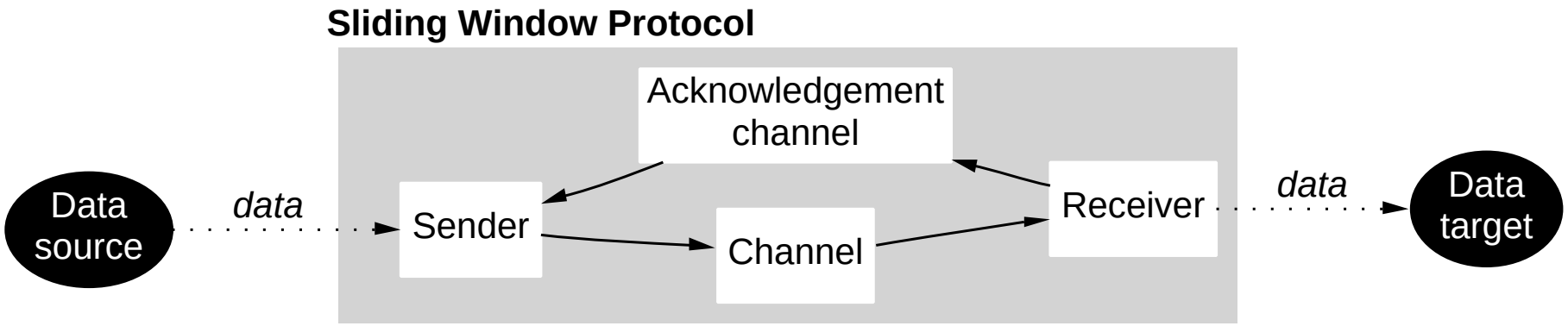
Interfaces to Other Tools (3/3)

MARIA supports two fundamentally different output formats.

Models can be unfolded to low-level Petri nets in the native format of LOLA, PEP and PROD. The unfolded models can then be analysed with these tools, which at the moment support better state space reduction methods than MARIA does.

State spaces of models, or selected parts of state spaces can be exported as directed graphs in the GraphViz format, or as labelled state transition systems in TVT format.

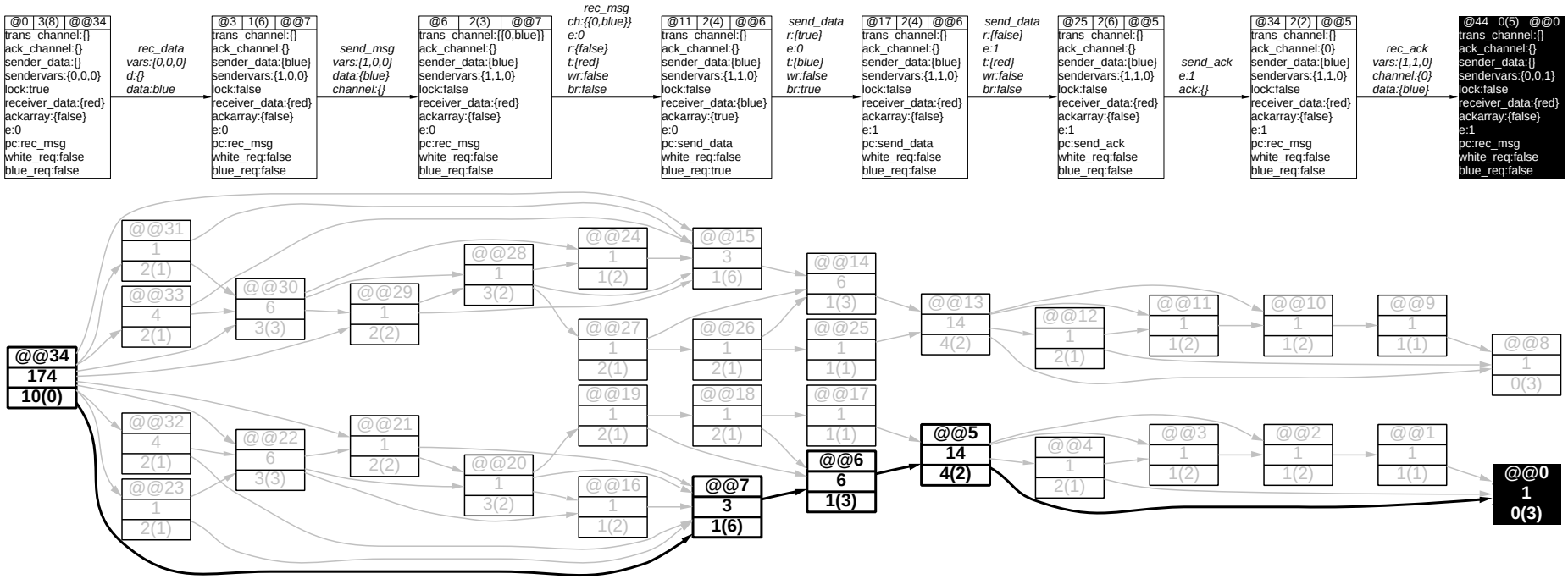
Example: Verifying a Sliding Window Protocol (1/3)



We have constructed a model of the protocol in the MARIA modelling language (roughly 200 lines of text). In this variation, two kind of messages are sent until a third kind of message ends the transmission. The sender and the receiver have a buffer for 1 message. The reachability graph contains 264 states and 582 arcs. If the transmission was continuous, the initial state would be reachable from each reachable state, and the reachability graph would thus have one *strongly connected component*.

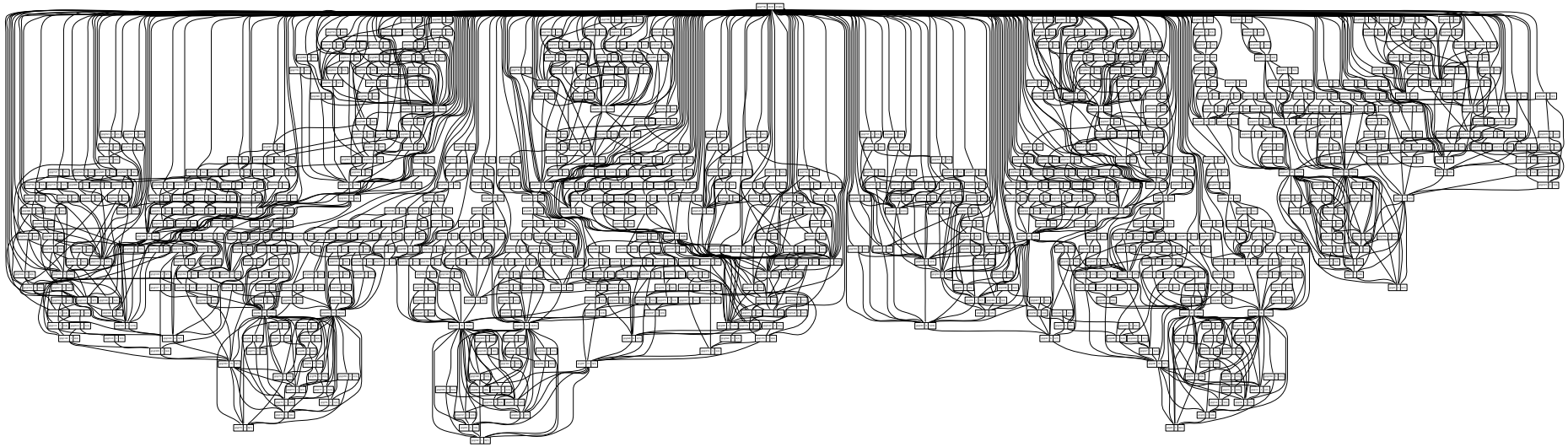
Example: Verifying a Sliding Window Protocol (2/3)

Because the model deadlocks once the terminating message has been sent, its reachability graph has several strongly connected components:



Example: Verifying a Sliding Window Protocol (3/3)

Growing the maximum length of the buffers grows the state space exponentially. With sending and receiving buffers for 2 messages, there are 8,268 states and 25,168 events. The component graph has 737 nodes and 2,048 arcs.



The Performance of MARIA

It does not make sense to try to present the state spaces MARIA can handle with one picture. There is no need to display all the states of the system if we are only interested in one execution path that demonstrates erroneous behaviour. Since MARIA is also an interactive tool, it is easy to analyse the results.

The biggest system analysed thus far is a simplified model of a radio link control protocol. Its reachability graph consists of 15,866,988 states and 61,156,129 events. The analysis employed a 266 MHz Pentium II system for 10 hours and occupied 1.55 GB of disk space but only 5 MB of memory. (The compilation option that allows memory-mapped access to reachability graph files had not been implemented then.)

The biggest part of the disk space was needed for representing the values of transition variables. Even though the model made use of complex structured data types, MARIA used only 40–50 bytes for each state and a bit over 10 bytes for each arc.

Properties that Can Be Verified (1/2)

A fully automated analysis can detect 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 *safety* and *liveness properties* e.g. with formulae of temporal or modal logic.

If the analyser detects a violation of the desired properties, it displays a *counterexample*, a violating chain of events that is possible in the system.

Properties that Can Be Verified (2/2)

An execution that violates a safety property (“nothing bad happens”) is an event chain from the initial state of the system to a reachable “bad” state.

A liveness property (“something good eventually happens”) is violated if the system can infinitely execute a loop of events without performing a “good” event. In one sense, it is comparable to a performance requirement: useful computations have to proceed in some finite time. Our formalism cannot express exact time.

Safety properties are much easier to check than liveness properties. Usually this kind of check is combined with *probabilistic verification*. A probabilistic verification run covers the whole state space with some probability. Repeating such runs reduces the probability that some execution paths remain unexplored.

Applicability (1/2)

The motivation behind MARIA was to create a reachability analyser and a model checker for a language whose expressive power is close to high-level programming and specification languages (such as C++ and SDL). Powerful operations make it possible to model complex communications without introducing superfluous intermediate states.

Users need not know the formalism of our analyser. They use the language of their own application field, and a *application-specific front-end* provides an interface

- by translating user programs or specifications to the internal formalism,
- by allowing the desired properties to be described in the application language, and
- by displaying erroneous behaviours as execution diagrams of the application.

Applicability (2/2)

Using a generic formalism has some advantages when compared to application specific formalisms. Implementing more powerful analysis methods immediately benefits all languages whose translation to the modelling language has been automated.

At the moment we have experimental front-ends for the Specification and Description Language standardised by ITU-T, and for its variant developed at Nokia, TNSDL. Thanks to the data type system of MARIA, it is easy to translate expressions and message passing.

We are working on a Java front-end that is based on the Bandera framework developed at the SAnToS laboratory of Kansas State University, USA, which in turn is based on the SOOT tools of the Sable group at McGill University, Quebec, Canada. It will be interesting to compare the results with other model checkers that Bandera supports.

MARIA is also extremely suitable for modelling systems manually.

Availability

MARIA can be freely used by anyone under the same terms as e.g. the Linux operating system, since it is covered by the General Public License published by the GNU project. We believe that in this way, the threshold of getting familiar with the tool and extending it becomes lower.

Our analyser has been developed in the UNIX environment. Part of it works in any system for which a standard-compliant C++ compiler is available.

If you got interested, visit our home pages at the address

`http://www.tcs.hut.fi/maria/`

and contact us. We are constantly looking for interesting systems whose modelling and analysis help us to develop our tool.