LoLA
A Low Level Analyser

Karsten Schmidt

Humboldt–Universität zu Berlin
kschmidt@informatik.hu--berlin.de

Abstract. With LoLA, we put recently developed state space oriented algorithms to other tool developers disposal. Providing a simple interface was a major design goal such that it is as easy as possible to integrate LoLA into tools of different application domains. LoLA supports place/transition nets. Implemented verification techniques cover standard properties (liveness, reversibility, boundedness, reachability, dead transitions, deadlocks, home states) as well as satisfiability of state predicates and CTL model checking. For satisfiability, both exhaustive search and heuristically goal oriented system execution are supported. For state space reduction, LoLA features symmetries, stubborn sets, and coverability graphs.

1 General Rer

coverbility graph

The development of LoLA started in the end of 1998 as an experimental implementation of several state space reduction techniques. Meanwhile, a data stream oriented interface has been added that allows other tools to exchange data with LoLA.

In principle, LoLA can run stand-alone (with a text editor for creating input). However, interface design was guided by the idea to run LoLA as a background service of another tool, invisible to the user. Therefore, LoLA does not have a graphical interface nor does it have any features that support modelling (such as hierarchies, modules, or version control).

We see the following advantages of a pure verification-only tool running in the background of a domain-specific tool:

- Developers of application oriented tools do not need to implement general-purpose verification techniques by their own;
- it is easier to keep pace with changes such as further improvements, optimisations, or new techniques (several verification techniques share the same pattern of communication: input system description and specification of the property to be verified, output verification result yes/no/don't know and — if available — a witness or counterexample execution);
- publications of verification techniques do not always report all details, possible shortcuts and optimisations of the presented techniques;

M. Nielsen, D. Simpson (Eds.): ICATPN 2000, LNCS 1825, pp. 465–474, 2000.

- it is easier to maintain parallel execution of different techniques (or the same technique with different parameters) in competition — this is particularly interesting in the face of well known discrepancies between worst case complexity and "typical case" run time ("typical" usually depends on the verification technique).

Example. In 1994, we reported an algorithm that is able to calculate symmetries of place/transition nets [4] and implemented the technique in the tool INA[1](Humboldt-Universität zu Berlin). Other tool developers implemented this algorithm, too, and reported some optimisations they had found. It turned out that most of these optimisations were covered by our own INA implementation as well (though not mentioned in our report). Thus, double work was done concerning both implementation and further tuning.

In 1997 and 1999, we published significant improvements to the symmetry algorithm, both concerning run time and memory efficiency. Had we distributed our technique in LoLA style already in 1994, it would be easy for the above mentioned developers to participate in our improvements just by downloading a new LoLA version. Without a service like LoLA, they either need to keep the old, less powerful algorithm or have to implement the new technique again.

Integrating the existing INA implementation into other tools (as done with the PEP tool (University of Hildesheim)) turned out to be a costly task since INA's interface was first and foremost designed for interactive use which is difficult to be simulated by machines. For creating LoLA, providing simple patterns for communication with other tools was consequently a major criterion.

We would like to notice that there are several tools that are prepared to integrate external services or actually do so. Among these tools are PEP (University Hildesheim/University Oldenburg), CPN-AMI (Université Paris VI), and the Petri Net Kernel (Humboldt–Universität zu Berlin).

As we understand LoLA as a service for other tools, LoLA itself does not have features that are specific for particular application domains. Input consists of plain and unstructured place/transition nets and parameters to the verification task specified at compile time (for instance, the name of a transition to be checked for liveness). Output consists of the answer to the verification query (a value ranging between "yes", "no", and "no answer") and, if existing, a witness or counterexample execution proving the given answer. On demand, additional output such as the system symmetries or the produced state space can be generated, too.

Benefits from LoLA's integration depend on the question whether there are verification problems that LoLA is able to solve and which are important for

[1] references to all tools mentioned in this paper can be found through the Petri net database that is accessible from the Petri net home page `http://www.daimi.au.dk/PetriNets/`

the host tool's application domain. LoLA supports the exploration of deadlocks (dead states), liveness, boundedness, reversibility, home states, reachability of states or state properties, and properties that are expressible in a simple branching time temporal logic.

Additionally, it is crucial whether LoLA can cope with typical system sizes of that domain. For the latter condition, it is difficult to give any prediction since the power of most state space reduction techniques implemented in LoLA varies from "amazing" to "almost none", depending on the particular system under verification. We have tested nets with about 100 nodes where LoLA failed for some verification task while we have seen other nets with 90000 nodes where LoLA was successful.

2 How to Use LoLA

LoLA distribution contains a configuration file. By editing this file, the user can customise the verification technique he or she wants to use. Customisation consists of fixing the property to be analysed, selecting the reduction techniques to be applied, choosing between breadth-first and depth-first strategy (use of breadth-first strategy is crucial for preserving shortest paths). Furthermore, capacities of certain data structures can be customised. Fig. 1 shows a part of the configuration file.

Having fixed a configuration, a compiler run generates an executable program. The resulting program reads a place/transition net description from a file or data stream, computes (if specified) the net symmetries, generates a (reduced) reachability graph and outputs the answer to the specified analysis query. The amount of output information (target state, witness path, state space, computed symmetries) can be controlled by command line options. Additionally, statistical information (such as number of states and edges of the generated graph, number of computed symmetries) is reported.

LoLA expects net descriptions in programming language style (see figure 2). In similar style, additional information about the verification task can be passed to LoLA. Each kind of LoLA output is preceded by a similar keyword identifying it as witness path, final state or whatever. Beyond passing net and task dependent parameters to LoLA and scanning LoLA results, no communication is necessary to control LoLA. Input and output languages follow regular (Chomsky type 3) grammars documented in the LoLA manual.

LoLA regularly produces messages in order to reveal its own progress. Frequency of these messages is customisable.

The style of using LoLA is partly comparable with INA and partly with PROD. In both INA and LoLA, the net description can be passed through a data file (though INA's language is much harder to read). However, in INA the task to be executed must be chosen through an interactive dialogue which is rather case sensitive and is responsible of most integration problems with INA. Additionally, pre–compiled configuration of the verification task has the advantage that code not relevant to the chosen task is simply not present in

```
//#define COVER
#define STUBBORN
#define SYMMETRY

#define GRAPH depth_first
//#define GRAPH breadth_first


//#define REACHABILITY
//#define MODELCHECKING
//#define BOUNDEDPLACE
//#define BOUNDEDNET
#define DEADTRANSITION
//#define REVERSIBILITY
//#define HOME
//#define FINDPATH
//#define FULL
```

Fig. 1. Part of LoLA's configuration file. Lines starting with `//` are comments and thus considered not present. In the sketched version, the resulting program would expect a net and a name of a transition t. Then it would start constructing a symmetrically and stubborn reduced reachability graph using depth first strategy. Both the used symmetry group and the version of stubborn sets would assure that death of t is preserved by the reduction. As soon as a state enabling t is found, the program returns with a path from the initial marking to that state. Due to depth first exploration this path is not necessarily a shortest one. If there is no state enabling t, the program returns after exhaustive exploration of the reduced graph.

the executable version. In INA, there is a lot of statements like `IF strategy = depth_first ...`. The evaluation of such statements costs only little, but inside a frequently executed loop this time sums up to a significant amount. This is one of the reasons why LoLA constantly outperforms INA on comparable tasks.

In PROD, both verification task and net description are subject to the compiler run. This way, additional speed-ups can be achieved. On the other hand, changes in the net require new compiler runs. Thus, an integrating tool needs to control the generation process of the PROD executable program during it's own run.

Concerning LoLA, we think that pre-compiled versions for at most a handful of configurations (of 350 KByte each) are sufficient for any host tool such that at run time of a host tool neither compilation (as in PROD) nor simulating an interactive session (as for INA) are necessary at the time the host tool is running.

```
PLACE eating0, eating1,  ( ..... )
      thinking2, thinking3, thinking4;

MARKING thinking0 : 1, thinking1 : 1, ( ..... ), fork4 : 1;

TRANSITION releaseleft0
CONSUME eating0: 1;
PRODUCE hasright0: 1, fork0: 1;

( ......)

TRANSITION takeright4
CONSUME hasleft4: 1, fork0: 1;
PRODUCE eating4: 1;
```

Fig. 2. Sketch of a LoLA input file for a dining philosophers net

3 State Space Exploration Techniques in LoLA

LoLA is able to compute the symmetries of a place/transition net automatically using the method presented in [5] and to calculate a symmetrically reduced reachability or coverability graph. For the integration of the symmetries into reachability graph generation, the user can choose between three different strategies. This choice allows the user to use the optimal method for a particular case (advantages and disadvantages of the strategies are discussed in [6]). LoLA always uses the largest possible symmetry group (which leads to the most condensed state space) that is capable of preserving the property to be verified. For model checking and home state verification, symmetries are switched off.

With or without symmetries, stubborn set techniques [9,10]) can be applied. This way, reduced state spaces that preserve a particular property can be generated. The list of analysable properties includes reachability (of a full marking or a state predicate), boundedness (of the net or a particular place), dead transitions, home states, and reversibility. If possible, this list will be extended. Symmetry and stubborn set methods can be applied jointly. The user can switch between different graph exploration strategies (depth first, breadth first). When using the latter strategy, shortest paths can be calculated for reachability related properties (including reachability of predicates and dead transitions). LoLA always selects a version of stubborn sets that preserves the property under verification (for standard properties, the versions are discussed in [8]).

A model checker based on the ALMC algorithm [11] is implemented and can be used with stubborn set reduction. Thereby, AG and EF operators are bound to stubborn set reduction as described in [7] while the remaining operators rely on the reduction technique of [2].

The size of systems that can be handled by LoLA depends on the reduction power of the symmetry and stubborn set techniques for the particular system and verification problem. Especially stubborn set reduction power is rather difficult

```
# lola ph500.net -S
2500 Places
2000 Transitions
 computing symmetries...
499 generators in 1 groups for 500 symmetries found.
0 dead branches entered during calculation.
>>>>> 1499 States, 2496Edges, 1499 Hash table entries
>>>>> 1 States, 0Edges, 1 Hash table entries
not reversible: no return to m0 from reported state
STATE
hasleft0 : 1,
hasleft1 : 1,
(...)
hasleft499 : 1
#
```

Fig. 3. LoLA execution in a configuration that checks reversibility using symmetrically and stubborn reduced graph. `ph500.net` is a file containing a net for a 500 dining philosophers system. Option -S signals that a witness state be reported. LoLA reports the size of the net and the number of found symmetries. In pass 1 of the reversibility check, LoLA calculates 1499 states for finding representatives of all terminal strongly connected components. In pass 2, it checks for theses representatives whether it is possible to return to the initial marking. This check fails for the reported ("witness") marking which is the well known dead state (all philosophers have taken the fork to their left).

to predict. LoLA has successfully analysed the existence of a deadlock for the 1000 dining philosophers net (9000 nodes, 3^{1000} states) using a symmetrically and stubborn reduced graph of about 3000 nodes. On the other hand, it has failed for some much smaller nets due to the too large number of states.

4 Attracted Simulation

LoLA can be used for testing whether a state satisfying a given state predicate is reachable. Testing is based on simulation runs where only the simulation path but not the intermediate states are stored. Selection of the transition to be fired in a state is based on a heuristics that is based on stubborn set techniques and hash table hit statistics. In several cases, the heuristics proved its capability to attract simulation towards a state satisfying the given predicate (if such a state is reachable). For instance, LoLA has found a satisfying state for some system and some property during the first simulation run though there was only one such state among 3^{10000} which was several 10000 transition occurrences away from the initial state.

If simulation runs into a deadlock or exceeds a user defined limit of path length, LoLA resets to the initial state and starts new runs until the program is terminated.

If a state satisfying the predicate is found, LoLA can output a path leading to that state. In contrast to witness paths arising from state space exploration, this path may contain cycles. Nevertheless, according to our experience, the witness path produced by attracted simulation tends to be rather short.

Attracted simulation is between 20 times (net with less then 100 nodes) and 1000 times (nets with several 1000 nodes) faster than traditional state space exploration (concerning the number of transition occurrences per time unit) and much more memory–efficient (only initial and current state as well as the simulation path need to be stored). Attracted simulation provides only semi decision of the tested property. However, its possible answers (yes, reachable/don't know) are dual to linear programming techniques based on the state equation (no, not reachable/don't know).

5 Implementation Details

LoLA's architecture is rather simple. It consists of a parser for reading net description and verification information, a state space depository, and two search procedures — one for depth-first, one for breadth first search. Depth first search is enhanced by Tarjan's algorithm to detect strongly connected components in the state space on the fly. Depending on the user configuration, there are compiler directives that include or exclude specific code for applying selected reduction techniques and for instantiating search to the configured verification task.

5.1 State Space Management

For storing states, LoLA uses a hybrid data structure. The first state is stored as a simple vector. The more states one includes, the more the structure converges to a decision tree. Atop of this structure there is a hash table, i.e. every hash class is stored as a separate tree. Parameters of the hash function are randomly chosen. However, if symmetries are applied, the parameters are adapted such that that the hash function becomes symmetry respecting.

With this structure of the marking depository, containment of a marking can be decided in linear time. Furthermore, tree like structures have advantages when symmetries are used [6]. For the array kind of storage, the user can control the number of bits to represent the token count of a place. This way, LoLA can be customised for safe nets such that a 32 bit word can store the marking of 32 places. Graph information is separated from the search structure. A reachability graph node stores information about successor and parent states, dfs and min numbers (for detection of strongly connected components), a link into the marking depository, and (conditionally) information related to the analysis query.

5.2 Firing

Firing transitions is the core activity in state space generation. Thus, is it particularly important to spend as few as possible run time for this task.

We apply the single transition firing rule. We have minimised the number of enabledness checks. After having fired a transition, only transitions where enabledness could have changed are re–investigated. If, for example, transition $t1$ is fired in the situation depicted in figure 4, only for $t2$ and $t3$ an enabledness check would be forced while the remaining transitions retain their values from the previous state. Backtracking is organised by firing transitions backwards.

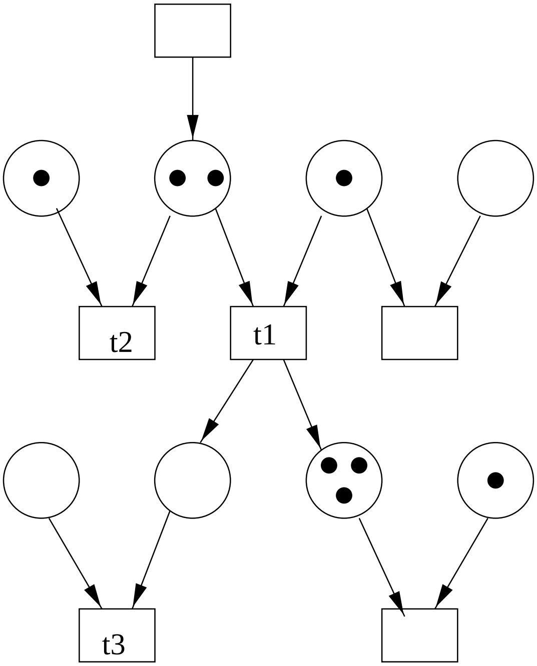

Fig. 4. Enabledness check in LoLA

5.3 Reduction Techniques

All reduction techniques are implemented as modifications to LoLA's central search procedures.

Stubborn set reduction is implemented as a modification of the routine that produces the list of transitions to be fired at a state. Stubborn sets are calculated using a closure operation on a property–dependent starting set. Reversibility and home states are checked using a two–phased procedure. In the first phase, representatives of all terminal strongly connected components of the reachability graph are computed using terminal component preserving stubborn sets. In the second phase, the property is verified by checking mutual reachability of these representatives using reachability preserving stubborn sets. For terminal component preserving stubborn sets as well as deadlock preserving stubborn sets, LoLA calculates minimal stubborn sets with respect to set inclusion.

Symmetric reduction influences the search of markings in the depository. Symmetry calculation and symmetry search is implemented exactly as described in [5,6]. If symmetries are used, LoLA's hash function will be symmetry respecting.

For coverability graph generation, a backward search for a covered state is performed. That slows down generation speed significantly. LoLA produces coverability graphs in the Karp–Miller–style [3] rather than in the Finkel–style [1]. The latter would require to remove already calculated states whereas the data structure of the marking depository supports only search and insertion as efficient operations.

Since all reduction techniques concern different parts of reachability graph generation procedure, they can be switched on or off independently. However, LoLA does not allow the application of reduction techniques that do not preserve the chosen target property.

6 Performance

On a 400MHz Pentium II PC (LINUX), reachability graph generation speed (without reduction) ranges between 25000 states/sec (net with 20 places) and 700 states/sec (net with 5000 places). With stubborn reduction, we still obtain up to 16000 states/sec (small net) or 400 states/sec (large net). Attracted simulation reaches about 500000 transition occurrences per second independently of the net size.

Symmetry calculation for a net with a few thousand nodes requires usually 3 to 5 minutes. The speed of producing a symmetrically reduced graph varies significantly, depending on the applied integration technique and the structure of the symmetry group.

For more detailed results concerning run–time and space consumptions when using reduction techniques, we refer the reader to the quoted papers presenting these techniques. The data reported there were all achieved using LoLA.

7 Availability

Under WWW page

`http://www.informatik.hu-berlin.de/~kschmidt/lola.html`

a free download as well as an online manual can be found.

LoLA runs on virtually all UNIX platforms (we have tested SOLARIS, LINUX and SUN-OS) as well as under WINDOWS-NT (using CYGNUS). If other platforms provide a C++ compiler and the GNU compiler generation tools (`bison` and `flex`), we do not expect problems for using LoLA there.

8 Future Work

We are going to enrich LoLA by more verification techniques, particularly in the area of efficient semi-decision procedures. Then, we plan to provide a platform that allows competition parallel run of several instances of LoLA (in different configurations) or even distributed state space generation in workstation or PC networks.

References

1. A. Finkel. A minimal coverability graph for Petri nets. *Proc. 11th International Conference on Application and Theory of Petri nets*, pages 1–21, 1990.
2. R. Gerth, R. Kuiper, D. Peled, and W. Penczek. A partial order approach to branching time logic model checking. *3rd Israel Symp. on the Theory of Computing and Systems, IEEE 1995*, pages 130–140, 1995.
3. R. M. Karp and R. E. Miller. Parallel programm schemata. *Journ. Computer and System Sciences 4*, pages 147–195, 1969.
4. K. Schmidt. Symmetries of Petri nets. *Informatik-Bericht* 33, Humboldt-Universität zu Berlin, 1994.
5. K. Schmidt. How to calculate symmetries of Petri nets. *Acta Informatica* 36, pages 545–590, 2000.
6. K. Schmidt. Integrating low level symmetries into reachability analysis. *Proc. 6th International Conference Tools and Algorithms for the Construction and Analysis of Systems*, LNCS 1785, pages 315–330, 2000.
7. K. Schmidt. Stubborn sets for model checking the EF/AG fragment of CTL. to appear in FUNDAMENTA INFORMATICAE, 2000.
8. K. Schmidt. Stubborn sets for standard properties. *20th International Conference on Application and Theory of Petri nets,* LNCS 1639, pages 46–65, 1999.
9. A. Valmari. Error detection by reduced reachability graph generation. *Proc. 9th European Workshop on Application and Theory of Petri Nets,* 1988.
10. A. Valmari. State of the art report: Stubborn sets. *Petri net Newsletter 46*, pages 6–14, 1994.
11. B. Vergauwen and J. Lewi. A linear local model checking algorithm for ctl. *Proc. CONCUR, LNCS 715*, pages 447–461, 1993.