LiQuor: A tool for Qualitative and Quantitative Linear Time analysis of Reactive Systems

Frank Ciesinski [*], Christel Baier
Universität Bonn, Institut für Informatik I, Germany

Abstract

LiQuor *is a tool for verifying probabilistic reactive systems modelled* Probmela *programs, which are terms of a probabilistic guarded command language with an operational semantics based on (finite) Markov decision processes.* LiQuor *provides the facility to perform a qualitative or quantitative analysis for ω-regular linear time properties by means of automata-based model checking algorithms.*

1 Introduction

State-transition-graphs with probabilistic and nondeterministic branching (Markov decision processes) arise naturally as operational models for probabilistic reactive systems, such as distributed systems with randomized coordination mechanisms, communication protocols that operate on unreliable channels or protocols for processes where stochastic assumptions about the frequency of interactions with the environment (e.g. a user) are available. Formal verification of probabilistic reactive systems typically requires checking qualitative and quantitative properties. The former means checking whether a certain path-event holds almost surely under all schedulers, while the task of a quantitative analysis is to check whether a given (lower or upper) probability bound for a certain path-event can be guaranteed for all scheduling policies.

The model checking tool LiQuor, short for "Linear time Qualitative/Quantitative analysis of reactive systems", serves to compute minimal or maximal probabilities for ω-regular path-properties of a probabilistic reactive systems. E.g. it can be used to verify for a randomized leader election algorithm that the probability to find a leader within 5 rounds is bounded below by 0.095 for any scheduling policy. Special algorithms have been implemented to treat qualitative properties stating that a linear time property holds almost surely. The modelling language, called Probmela [BCG04], is a dialect of SPIN's input language Promela (see e.g. [Hol03]).

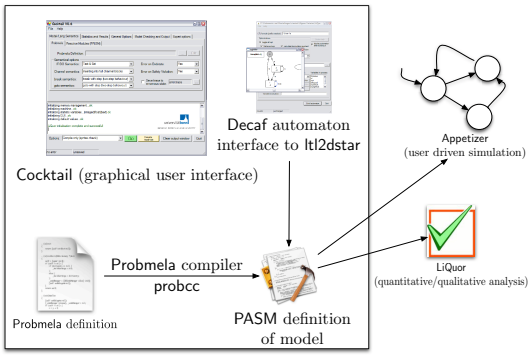

Figure 1. Organization of LiQuor components

2 Current features of LiQuor

Input language and internal representation of the system model. LiQuor's input language Probmela provides a simple and intuitive formalism to repesent the behavior of the processes that run in parallel and might communicate over shared variables, by CSP-like handshaking over synchronous channnels or by asynchronous message passing over FIFO-channels. Probmela provides standard concepts of imperative programming languages, such as conditional or repetitive commands with (possibly overlapping) boolean guards, deterministic and randomized assignments, a probabilistic choice operator, and communication actions. The given Probmela-program $\mathcal{P}$ is first compiled into an XML-based intermediate language, called PASM, which specifies the MDP-semantics $\mathcal{M}$ of $\mathcal{P}$ by means of a bytecode-based JAVA-like virtual machine and serves as starting point for the DFS-based on-the-fly construction of the reachable fragment of the system to be analyzed. The user can use a GUI-based application (Cocktail) to specify various model checking parameters and initiate the translation between Probmela and PASM (see fig. 2). It is also possible to simulate the model in a single step manner using the (also GUI based) application Appetizer. During simulation data like the current variable evaluations and the program location is displayed.

[*]The author is supported by the DFG-NWO-project "VOSS II".

Implemented model checking algorithms. LiQuor relies on the automata-based approach to model check linear time properties [VW86, CY95, dA97]. The given LTL-formula ϕ is translated into a deterministic Rabin- or Streett automaton $\mathcal{A}_\phi$ using the tool ltl2dstar [KB05] which is linked with LiQuor via the graphical interface Decaf. The latter provides a PASM-representation of $\mathcal{A}_\phi$. Using the PASM-code for the Probmela-program and the automaton, LiQuor then applies an DFS-based construction of the product-MDP $\mathcal{M} \times \mathcal{A}_\phi$ and calculates its end components. These are strongly connected sub-MDPs such that, once entered an end component C, a scheduler can enforce the computations to stay inside C forever and to visit all its states infinitely often almost surely. To check Rabin and Streett acceptance in a qualitative setting it suffices to analyze the end components by means of graph algorithms. For the quantitative analysis, the maximal or minimal probabilities for reaching an accepting end component are computed by means of a linear program. In LiQuor, the computation of the (maximal) end components relies on an iterative calculation of strongly connected components (SCCs) [CY95, dA97] with a variant of Tarjan's SCC-algorithm. Although the worst-case complexity is quadratic in the size of the product-MDP, our experimental studies show that in most cases only a few iterations are sufficient. For the qualitative analysis where only the existence or non-existence of a reachable accepting end component has to be checked, several heuristics are implemented in LiQuor that allow for "early termination" (i.e., abortion before the computation of all accepting maximal end components is completed). The linear programs required for the quantitative analysis are solved in LiQuor either by the Simplex-algorithm (using the external tool lp-solve) or by an iterative algorithm in the style of value iteration. The latter turned out to be much faster and more appropriate to handle large state spaces.

3 An example case study

Several case studies have been performed with LiQuor to analyse, e.g., randomized mutual exclusion and leader election algorithms. We briefly report here on an example that concerns the UMTS synchronization procedure and has been addressed in cooperation with a major telephone company. The service structure in UMTS is divided in *domains*. When the user (e.g. by using an UMTS enabled mobile phone) uses a domain (i.e. makes a voice phone call *or* initiates an internet connection) the phone uses a *temporal key* for authentification to the network. These keys are obtained from the network in a bundle of several keys (called *batch*) when needed and are - among much other parameters for encryption and safety purposes - equipped with a timestamp (i.e. a precise clock value). Each key is for one usage in it's domain only and stored after usage in a joint buffer (key table). The UMTS protocol requires that only keys are used that are with respect to the attached timestamp *newer* (the keyword for UMTS experts here is *key freshness*) than keys that are already stored (and therefore used) in the buffer. Because of details the buffer protocol and because of the independent key generation mechanisms for each domain it is possible that a key is to be used that is (with respect to the attached timestamp) *older* than a key inside the used-buffer and therefore violates the freshness condition. This case is called *synchronization-failure* and requires jettisoning all unused keys (because it is assumed that they all violate the freshness condition as well) and obtain a batch of new keys. It would be undesirable for the network operator of this situation occurs very often. We specified all relevant protocol features using Probmela and used probabilities from the provider's database to specify assumptions about the user behaviour. Some basic results are given in tables (see figures 2 and 3).

slots	rounds	states	constr.	iter.	time	p_{error}
10	30	151885	15157	546	85s	0.0255851
20	40	182835	23397	768	141s	$4,2 \cdot 10^{-6}$
32	50	220445	227556	1077	216s	$1,45 \cdot 10^{-8}$

Figure 2. results for formula $\diamond error$

slots	states	time	p_{fail}
10	125775	13s	1
20	627295	99s	1

Figure 3. formula $\Box \diamond error$

References

[BCG04] C. Baier, F. Ciesinski, and M. Größer. Probmela: a modeling language for communicating probabilistic systems. In *Proc. of the Second ACM-IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE)*, pages 57–66. IEEE CS Press, 2004.

[CY95] C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *Journal of the ACM*, 42(4):857–907, 1995.

[dA97] L. de Alfaro. *Formal Verification of Probabilistic Systems*. PhD thesis, Stanford University, Department of Computer Science, 1997.

[Hol03] G. Holzmann. *The SPIN Model Checker, Primer and Reference Manual*. Addison Wesley, 2003.

[KB05] Joachim Klein and Christel Baier. Experiments with deterministic ω-automata for formulas of linear temporal logic. volume 3845, pages 199–212, 2005.

[VW86] M. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification (preliminary report). *Proc. LICS'86*, pages 332–344, 1986.