TLQ: A Query Solver for States

Mihaela Gheorghiu and Arie Gurfinkel

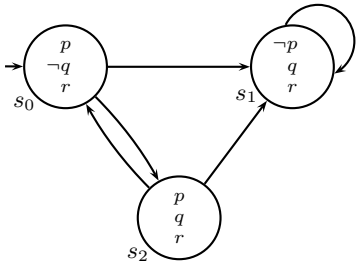
Department of Computer Science, University of Toronto,
Toronto, ON M5S 3G4, Canada.
Email: {mg,arie}@cs.toronto.edu

Abstract. In this paper, we present TLQ, a tool that finds the state solutions to any CTL query. It extends existing approaches specialized in finding state solutions to special kinds of queries, being at the same time more efficient than general query checkers that can find all solution to any query. Its generality allows its application to new problems, such as finding stable states, as required in some applications from genetics. We describe the implementation of TLQ on top of the model-checker NuSMV, and show its effectiveness in finding the stable states of a gene network.

We describe our tool TLQ that applies to the analysis of state-transition models, and can answer questions of the type: "What are the states that satisfy a property φ?". We address those cases where φ cannot be formulated in a temporal logic appropriate for model checking. Instead, φ can be formulated as a temporal query suitable for query checking. An example of such a property φ is "reachability". We cannot formulate a CTL property whose model-checking results in those states reachable from a given state, but we can formulate the CTL query $EF\ ?$ that literally encodes the question: which states are reachable (from the initial state)? While most model-checkers routinely compute reachable states, they do so by employing specialized procedures. Our tool has been motivated by the need to answer similar questions in a more systematic way than is currently done.

Query-checking [5] is an extension of model-checking meant to *discover* the properties that hold in a model, by allowing unknowns in a given formula that is checked. A CTL query is a CTL formula with a missing propositional subformula, designated by a placeholder ("*?*"). A *solution* to the query is any propositional formula that, when substituted for the placeholder, results in a CTL formula that holds in the model. General query checkers find all propositional solutions of a given CTL query on a model. For example, consider the model in Figure 1(a), where each state is labeled by the atomic propositions that hold in it. A query $AF\ ?$ on this model asks "what propositional formulas eventually hold on all paths" and has solutions $p \wedge \neg q \wedge r$ and $q \wedge r$. General query checking requires a number of model-checking runs that is double-exponential in the number of variables in the model, which is prohibitive for realistic models. For this reason, current query checkers that find all propositional solutions are either restricted to certain CTL formulas to be efficient (same cost as one model checking run) [5], or unrestricted and thus inefficient [3, 6, 10].

For many applications, however, only state solutions to queries are needed. In our example, the solution $p \wedge \neg q \wedge r$ corresponds to the state s_0 and is a state solution,



	Model	Query	Algorithms (minutes) Ours	Naive
1	original	$EF\ AG\ ?$	117	145
2	mutant 1	$EF\ AG\ ?$	116	144
3	mutant 2	$EF\ AG\ ?$	117	145
4	mutant 3	$EF\ AG\ ?$	117	146
5	original	$AG\ ?$	116	145
6	original	$EF\ ?$	118	146
7	original	$AF\ ?$	117	145

Fig. 1. (a) Example state machine; (b) Experimental results.

whereas $q \wedge r$ corresponds to a set of states $\{s_1, s_2\}$ and neither s_1 nor s_2 is a solution by itself. Finding only the state solutions is an easier problem that can be solved with a single-exponential number of calls to a model checker. Our tool TLQ matches this complexity, is fully symbolic (consists of only one model-checking run), and unrestricted in the class of CTL queries that it handles.

Some potential applications of TLQ, beside finding reachable states, are: finding procedure summaries, or dominators/postdominators in program analysis [11, 2, 1]. We also show a new application coming from genetics where stable states of a gene network are sought, which amounts to finding the state solutions to CTL query $EF\ AG\ ?$. We show that TLQ solves this problem more efficiently than a naive algorithm.

Implementation. TLQ is based on the the symbolic query-checking framework described in [9]. The tool receives as input a CTL query and a model. Assuming a model with two variables p, q, first the $?$ is replaced by the expression

$$\begin{array}{l}(p \wedge q \wedge x \wedge y) \vee \\ (p \wedge \neg q \wedge x \wedge \neg y) \vee \\ (\neg p \wedge q \wedge \neg x \wedge y) \vee \\ (\neg p \wedge \neg q \wedge \neg x \wedge \neg y)\end{array}$$

The resulting CTL formula is then checked with NuSMV. The output shows the set of state solutions as a propositional formula over the variables x, y. It is easy to see how this approach generalizes to any number of variables. For $AF\ ?$ on the model in Figure 1(a), TLQ results in $x \wedge \neg y \wedge z$, corresponding to the only state solution $p \wedge \neg q \wedge r$. Variables $x, y, z, \ldots$ do not appear in the model being checked, and are "copies" of the model variables $p, q, r, \ldots$, respectively. More details on the theory behind TLQ can be found in [8].

Application. We applied TLQ to finding stable states of a model, and evaluated its performance by comparison to the naive approach to state query checking.

In a study published in plant research, a model of gene interaction has been proposed to compute the "stable states" of a system of genes [7]. This work defined stable states as reachable gene configurations that no longer change, and used discrete dynamical systems to find such states. A different publication, [4], advocated the use of

Kripke structures as appropriate models of biological systems, where model checking can answer some of the relevant questions about their behaviour. [4] also noted that query-checking might be useful as well, but did not report any applications of this technique. Motivated by [4], we repeated the study of [7] using our state query-checking approach.

The model of [7] consists of 15 genes, each with a "level of expression" that is either boolean (0 or 1), or ternary (0,1, or 2). The laws of interaction among genes have been established experimentally and are presented as logical tables. The model was translated into a NuSMV model with 15 variables, one per gene, of which 8 are boolean and the rest are ternary, turning the laws into NuSMV next-state relations. The model has 559,872 states and is available from `www.cs.toronto.edu/~mg/flower.smv`.

The problem of finding all stable states of the model amounts to finding the state solutions to the query $EFAG\ ?$. Performance of TLQ and its naive counterpart for this query is summarized in the top row of the table in Figure 1(b), where the times are reported in minutes. The naive algorithm was also implemented using NuSMV by generating all possible states, replacing each for the placeholder in $EFAG\ ?$ and calling NuSMV to check the resulting formulas. Both algorithms were run on a Pentium 4 processor with 2.8GHz and 1 GB of RAM. Our algorithm gave an answer in under two hours, being about 20% faster than the naive.

To have a larger basis of comparison between the two algorithms, we varied the model (see rows 2-4), and the checked queries (see rows 5-7). Each "mutant" was obtained by permanently switching a different gene off, as indicated in [7]. The performance gain of our algorithm remained unchanged.

Conclusions. While our preliminary experiments look promising, we expect that our tool requires additional heuristics before it can handle larger problems. One bottleneck of the implementation is the size of the data structures used, which is a well-known problem with symbolic algorithms. A source of heuristics may be the particular applications of the tool. Due to its generality, TLQ can be used wherever queries can be formulated whose state solutions are sufficient. Upon application, further domain knowledge may suggest ways to improve our general algorithm for the particular queries of interest. We envision TLQ being used rather as a template for the development of efficient techniques for particular problems. The tool is still under development, but a stable version is available upon request, as a patch to NuSMV.

References

1. B. Aminof, T. Ball, and O. Kupferman. "Reasoning About Systems with Transition Fairness". In *Proc. of LPAR'04*, volume 3452 of *LNCS*, pages 194–208, 2005.
2. T. Ball and S. Rajamani. "Bebop: A Symbolic Model Checker for Boolean Programs". In *Proc. of SPIN'00*, volume 1885 of *LNCS*, pages 113–130, 2000.
3. G. Bruns and P. Godefroid. "Temporal Logic Query-Checking". In *Proc. of LICS'01*, pages 409–417, 2001.
4. N. Chabrier-Rivier, M. Chiaverini, V. Danos, F. Fages, and V. Schachter. "Modeling and Querying Biomolecular Interaction Networks". *Theor. Comp. Sci.*, 325(1):25–44, 2004.
5. W. Chan. "Temporal-Logic Queries". In *Proc. of CAV'00*, volume 1855 of *LNCS*, pages 450–463, 2000.

6. M. Chechik and A. Gurfinkel. "TLQSolver: A Temporal Logic Query Checker". In *Proc. of CAV'03*, volume 2725 of *LNCS*, pages 210–214, 2003.
7. C. Espinosa-Soto, P. Padilla-Longoria, and E. R. Alvarez-Buylla. "A Gene Regulatory Network Model for Cell-Fate Determination during Arabidopsis thaliana Flower Development That Is Robust and Recovers Experimental Gene Expression Profiles". *The Plant Cell*, 16:2923–2939, 2004.
8. M. Gheorghiu, A. Gurfinkel, and M. Chechik. *Finding State Solutions to Temporal Logic Queries*. `www.cs.toronto.edu/~mg/state_qc.ps`.
9. A. Gurfinkel, M. Chechik, and B. Devereux. "Temporal Logic Query Checking: A Tool for Model Exploration". *IEEE Trans. on Soft. Engineering*, 29(10):898–914, 2003.
10. S. Hornus and P. Schnoebelen. "On Solving Temporal Logic Queries". In *Proc. of AMAST'02*, volume 2422 of *LNCS*, pages 163–177, 2002.
11. T. W. Reps, S. Horwitz, and M. Sagiv. "Precise Interprocedural Dataflow Analysis via Graph Reachability". In *Proc. of POPL'95*, pages 49–61, 1995.