GOAL: A Graphical Tool for Manipulating Büchi Automata and Temporal Formulae*

Yih-Kuen Tsay, Yu-Fang Chen, Ming-Hsien Tsai, Kang-Nien Wu, and Wen-Chin Chan

Department of Information Management, National Taiwan University, Taiwan

1 Introduction

In this paper, we present a tool named GOAL (an acronym derived from "**G**raphical Tool for **O**mega-**A**utomata and **L**ogics") whose main functions include (1) drawing and testing Büchi automata, (2) checking the language equivalence between two Büchi automata, (3) translating quantified propositional linear temporal logic (QPTL) formulae into equivalent Büchi automata, and (4) exporting Büchi automata as Promela code. The GOAL tool, available at `http://goal.im.ntu.edu.tw`, can be used for educational purposes, helping the user get a better understanding of how Büchi automata work and how they are related to linear temporal logics. It may also be used, as we shall explain below, to construct correct and smaller specification automata, supplementing model checkers that adopt the automata-theoretic approach, such as SPIN [5].

The automata-theoretic approach [11,1] to linear temporal logic model checking works as follows. Suppose A is the Büchi automaton modeling the system and B the Büchi automaton specifying a desired property. The problem of model checking translates into that of testing language containment $L(A) \subseteq L(B)$, which is equivalent to $L(A) \cap \overline{L(B)} = \emptyset$. As Büchi automata are closed under complementation and intersection, this reduces to testing if $L(A \times \overline{B}) = \emptyset$, namely the emptiness problem of Büchi automata. Because of the difficulty and high complexity in complementing a Büchi automaton, in practice, an automata-theoretic model checker typically assumes that the specification is given as a propositional linear temporal logic (PTL) formula. The model checker first negates a specification formula φ and then translates it into an automaton $B_{\neg\varphi}$ that represents all behaviors disallowed by φ, i.e., $L(B_{\neg\varphi}) = \overline{L(B_\varphi)}$ (where B_φ is a Büchi automaton equivalent to formula φ). Checking if $L(A) \cap \overline{L(B_\varphi)} = L(A \times \overline{B_\varphi}) = \emptyset$ is therefore the same as checking if $L(A \times B_{\neg\varphi}) = \emptyset$, where one only needs to construct the intersection (product) of A and $B_{\neg\varphi}$, and complementation is avoided.

Assuming that the specification is given as a PTL formula has two disadvantages. *First, it limits the type of properties that can be specified and checked.* An ideal automata-theoretic model checker would support some extension of PTL such as QPTL that is expressively equivalent to Büchi automata. The SPIN

* This work was supported in part by the National Science Council of Taiwan (R.O.C.) under grants NSC95-2221-E-002-127 and NSC95-3114-P-001-001-Y02 (iCAST).

O. Grumberg and M. Huth (Eds.): TACAS 2007, LNCS 4424, pp. 466–471, 2007.

model checker offers the user instead the possibility of directly defining $B_{\neg\varphi}$ in Promela. However, it provides no assist for the user to check the "correctness" of the defined automaton, i.e., if the automaton describes what is intended. Büchi automata are in general harder to get right than temporal formulae. *Second, the machine-translated automaton $B_{\neg\varphi}$ may be larger than an optimal and equivalent one.* Many algorithms exist for translating a PTL formula into an equivalent Büchi automaton, e.g., [3,4], but none of them guarantee optimality. As the emptiness checking of $A \times B_{\neg\varphi}$ requires time proportional to the size of the system automaton A and to that of the specification automaton $B_{\neg\varphi}$, a larger $B_{\neg\varphi}$ would mean a longer verification time. To reduce verification time, it may be worthwhile to construct a smaller $B_{\neg\varphi}$ manually. But again, a way for checking the correctness of a user-defined $B_{\neg\varphi}$ is needed.

This is one typical situation where the GOAL tool can be useful. First of all, GOAL is graphical, making a user-defined automaton easier for human inspection. More importantly, the correctness of a user-defined specification automaton can be checked against an easier-to-understand QPTL formula, by translating the specification formula into an equivalent automaton and testing the equivalence between the user-defined and the machine-translated automata. QPTL is expressively equivalent to Büchi automata [9]. GOAL also supports past temporal operators which make some specifications easier to write. In addition, GOAL provides a repository that contains common patterns of temporal formulae and their corresponding optimized and machine-checked Büchi automata. Once the specification automaton of an ideal size has been successfully constructed and checked, it can be exported as Promela code which can then be fed into the SPIN model checker.

GOAL was originally designed for learning/teaching Büchi automata and linear temporal logics. Despite the possibility of mechanical translation, a temporal formula and its equivalent Büchi automaton are two very different artifacts and their correspondence is not easy to grasp. Temporal formulae describe temporal dependency without explicit references to time points and are in general more abstract and easier to understand, while Büchi automata "localize" temporal dependency to relations between states and tend to be of lower level and harder to understand. Nonetheless, Büchi automata and their relation with linear temporal logics can be better understood by going through some translation algorithm with different input temporal formulae or simply by examining more examples of temporal formulae and their equivalent Büchi automata. This learning process, unfortunately, is tedious and prone to mistakes for the students, while preparing the material is very time-consuming for the instructor. Tool support is needed.

An earlier version of GOAL has been introduced and suggested for educational purposes in [10]. However, its inability in handling quantified temporal formulae limited the kind of Büchi automata that could be explored. It also lacked the exporting function that allows its use in combination with an automata-theoretic model checker. To the best of our knowledge, GOAL is the first graphical interactive tool for manipulating Büchi automata and temporal formulae that supports past temporal operators and quantification over propositions. There

are other tools that provide translation of temporal formulae into Büchi automata, e.g., LTL2BA [3]. However, none of them provide facilities for visually manipulating automata and the temporal logics they support are less expressive. The operations and tests on Büchi automata provided by GOAL are also more comprehensive than those by other tools.

2 Main Functions

Below is a brief description of the main functions of GOAL, followed by some implementation highlights.

- **Drawing and Testing Büchi Automata:** The user can easily point-and-click and drag-and-drop to create a Büchi automaton and test it. To get a feel of what kind of inputs the automaton accepts, the user can run it through some input words. More interestingly, an automaton can be tested for emptiness and two automata can be tested for language containment and equivalence, as well as simulation equivalence.
- **Checking the Language Equivalence between Two Büchi Automata:** This is a particularly useful test function. The equivalence test between two Büchi automata is built on top of the language containment test which in turns relies on the intersection and complementation operations and the emptiness test. If two automata are not equivalent, an infinite word which is contained in the difference of the two automata will be displayed as a counter example.
- **Translating QPTL Formulae into Equivalent Büchi Automata:** The user can type in a QPTL formula and ask GOAL to translate it into an equivalent Büchi automaton, as shown in Figure 1(a). Currently, GOAL imposes a restriction that a quantifier must not fall in the scope of a temporal operator. This restriction does not sacrifice expressiveness, as QPTL with the restriction is as expressive as the original unrestricted QPTL, which is expressively equivalent to Büchi automata [9]. Machine-translated Büchi automata are usually not optimal in terms of size, yet they are useful for verifying the correctness of user-defined automata (by the equivalence test). GOAL also supports past temporal operators which make some specifications easier to write, helping the user convey his intuition without much hacking.
- **The Automata Repository:** This repository contains a collection of frequently used QPTL formulae and their corresponding equivalent Büchi automata, which have been optimized by hand and checked by GOAL; see Figure 1(b) for an example.
- **Exporting Büchi Automata as Promela Code:** Once an automaton has been defined and tested, the user can export it in the Promela syntax on the screen or as a file, as shown in Figure 1(b). This makes it possible to use GOAL as a graphical specification definition frontend to an automata-theoretic model checker like SPIN.

GOAL is implemented in Java for the ease of installation. Its automata and graph modules were adapted from those of JFLAP [7], a tool for classic theory of computation. The most complicated algorithms in GOAL are those for

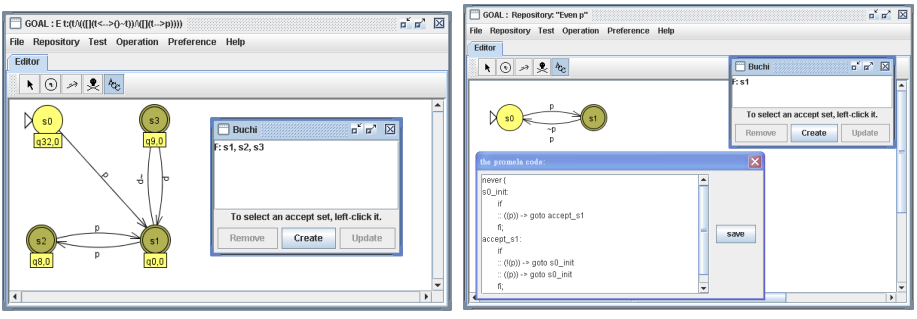

Fig. 1. Two equivalent Büchi automata that describe the property "p is true at every even position", which can also be expressed as a QPTL formula

translating temporal formulae into automata and for complementing automata. Our translation algorithm combines an adaptation of the tableau construction described in Manna and Pnueli's book [6] and the approach described in [9] for handling quantification. For automata complementation, we adopted the algorithm by Safra [8]. From inputs of a moderate size, these algorithms may produce very large automata, which are difficult to display and usually impossible to understand intuitively. However, this is not a serious problem, as on the one hand we intend GOAL to be used for educational purposes or for specification definition, where the input temporal formulae or automata tend to be small. On the other hand, the machine-generated automata are often used for equivalence tests, not for human inspection. Nonetheless, we did implement several methods for state reduction, for example, removing redundant states detected by simulation [2]. We have planned to include implementations of other translation and complementation algorithms, which would be useful for comparative studies.

3 Use Cases

We describe a number of use cases that illustrate how the GOAL functions may be combined and used in particular as a tool for learning/teaching Büchi automata and linear temporal logics or for specification development:

– **Checking correctness of a hand-drawn Büchi automaton:** Understanding a Büchi automaton is in general harder than understanding an equivalent temporal formula. Consequently, defining or drawing a Büchi automaton that conveys one's intention is also harder than writing a temporal formula for the same purpose. Whether a hand-drawn automaton is correct, i.e., if it conveys the specifier's intention, can be verified using GOAL by following these steps: (1) Write a QPTL (or PTL if it suffices) formula that specifies the same thing. (2) Translate the formula into an equivalent Büchi automaton. (3) Test the equivalence between the machine-translated and the hand-drawn automata. If the equivalence test is positive, then one can be assured that the hand-drawn Büchi automaton is indeed what is intended.

- **Manual optimization of a specification Büchi automaton:** In principle, a smaller specification automaton makes a model checker run faster. GOAL may be used to manually optimize a Büchi automaton by repeatedly merging or removing its states or transitions and checking if the resulting automaton is equivalent to a previous correct automaton. Though this is essentially a trial-and-error process, the equivalence test provided by GOAL will greatly ease the pain.
- **Understanding why PTL is strictly less expressive than Büchi automata:** The property "p is true at every even position" (an infinite word or sequence starts with position 0), or "Even p" for short, is a typical example for showing that PTL is strictly less expressive than Büchi automata. A plausible PTL formula for "Even p" would be "$p \wedge \Box(p \rightarrow \bigcirc\bigcirc p)$". Using GOAL, one can translate the formula into a Büchi automaton and compare it with the one for "Even p" from the repository. An equivalence test will show that the two automata are not equivalent and display a counter example. Indeed, the formula $p \wedge \Box(p \rightarrow \bigcirc\bigcirc p)$ is overly restrictive. Once p holds at some odd position, this formula forces p to hold at all subsequent odd positions, which is not required by "Even p". The property can, however, be expressed by a QPTL formula, e.g., $\exists t : t \wedge \Box(t \leftrightarrow \bigcirc\neg t) \wedge \Box(t \rightarrow p)$.
- **Combining GOAL with SPIN:** In the SPIN model checker, the specification can either be given as a PTL formula (without past operators) or directly as a Büchi automaton in Promela code. For a property that is not expressible in PTL, defining a suitable Büchi automaton becomes necessary. In this case, GOAL supplements SPIN by providing a convenient graphical interface for drawing and manipulating Büchi automata. Once the (negative) specification automaton of an ideal size has been successfully constructed and checked, it can be exported as Promela code. One can then copy-and-paste the Promela code to SPIN's model file as the "never claim" and continue the model checking procedure as usual.

4 Concluding Remarks

The GOAL tool will continue to be improved and extended. As the source of the acronym "GOAL" suggests, our long-term goal is for the tool to handle the common variants of omega-automata and the logics that are expressively equivalent to these automata. Currently, as by-products of Safra's complementation construction, GOAL already inlcudes Büchi to Rabin and Streett to Büchi translations. Although these variants of omega-automata do not necessarily have a direct impact on model-checking efficiency, they are powerful intermediaries for automata-based algorithms development. A tool that can visually manipulate these variants and perform their translations will be helpful in such developments. It is also of educational value, which should not be overlooked.

Acknowledgment. We thank Susan H. Rodger, the creator of JFLAP, at Duke University for granting us the permission to use and modify the JFLAP source code.

References

1. E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking*. The MIT Press, 1999.
2. K. Etessami and G. Holzmann. Optimizing Büchi automata. In *Proceedings of the 11th International Conference on Concurrency Theory (CONCUR 2000), LNCS 1877*, pages 153–167. Springer, 2000.
3. P. Gastin and D. Oddoux. Fast LTL to Büchi automata translation. In *Proceedings of the 13th International Conference on Computer-Aided Verification (CAV 2001), LNCS 2102*, pages 53–65. Springer, 2001.
4. R. Gerth, D. Peled, M.Y. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Protocol Specification, Testing, and Verification*, pages 3–18. Chapman & Hall, 1995.
5. G.J. Holzmann. *The SPIN Model Checker: Primer and Reference Manual*. Addison-Wesley, 2003.
6. Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safty*. Springer, 1995.
7. S. Rodger and T. Finley. JFLAP. http://www.jflap.org/.
8. S. Safra. On the complexity of ω-automta. In *Proceedings of the 29th Annual IEEE Symposium on Foundations of Computer Science (FOCS 1988)*, pages 319–327, 1988.
9. A.P. Sistla, M. Vardi, and P. Wolper. The complementation problem for Büchi automata with applications to temporal logic. *Theoretical Computer Science*, 49:217–237, 1987.
10. Y.-K. Tsay, Y.-F. Chen, and K.-N. Wu. Tool support for learning Büchi automata and linear temporal logic. Presented at the Formal Methods in the Teaching Lab Workshop, Hamilton, Canada, August 2006.
11. M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proceedings of the 1st Annual IEEE Symposium on Logic in Computer Science (LICS 1986)*, pages 332–344, 1986.