Tool Support for Learning Büchi Automata and Linear Temporal Logic[⋆]

Yih-Kuen Tsay, Yu-Fang Chen, and Kang-Nien Wu

Dept. of Information Management, National Taiwan University, TAIWAN

Abstract. Automata and logics are intimately related, and understanding their relation is instrumental in discovering algorithmic solutions to formal reasoning problems or simply in using those solutions. This applies to Büchi automata and linear temporal logic, which have become fundamental components of the model-checking approach to formal verification of concurrent systems. Translation of a propositional temporal formula into an equivalent Büchi automaton is routinely performed in many model-checking algorithms and tools. Albeit the possibility of mechanical translation, a temporal formula and its equivalent automaton appear to be two very different artifacts and their correspondence is not easy to grasp. In this paper, we introduce a graphical interactive tool, named GOAL, that can assist the user in understanding the relation between Büchi automata and linear temporal logic, and suggest possible usages and benefits of the tool in courses where model-checking techniques are covered. GOAL builds on the successful JFLAP tool for classic theory of automata and formal languages. One main function of GOAL is translation of a propositional temporal formula into an equivalent Büchi automaton that can be visually manipulated, for example, running the automaton on some input. GOAL also supports various standard operations and tests, including equivalence test, on Büchi automata. We believe that, with an easy access to temporal formulae and their graphically presented equivalent Büchi automata, the student's understanding of the two formalisms and their relation will be greatly enhanced.

1 Introduction

The model-checking approach to formal verification of concurrent systems seeks to automatically verify if the given system represented by an abstract model satisfies its specification. Because of its proven effectiveness and ease of use, model checking has become a viable alternative to simulation and testing in industry. Model checkers are also increasingly exploited by verification tools based on deductive (theorem proving) methods, as the work horses for decidable verification subtasks.

In one school of model checking, a concurrent system is equated semantically with a set of infinite computations and its desired behavioral properties are then specified in terms of those computations. The specification of a behavioral property typically asserts temporal dependency between occurrences of certain events (represented by propositions) and linear temporal logic has thus become a particularly popular class

[⋆] This work was supported in part by the National Science Council of Taiwan (R.O.C.) under grant NSC 94-2213-E-002-089.

of languages for specification. Temporal dependency between events may also be expressed with Büchi automata, which are finite automata operating on infinite words.

Indeed, automata and logics are intimately related, as we all have learned from classic theory of computation. Understanding their relation is instrumental in discovering algorithmic solutions to formal reasoning problems or simply in using those solutions. This applies to Büchi automata and linear temporal logic. It has been shown that Büchi automata and a variant of linear temporal logic called quantified propositional temporal logic are expressively equivalent. For the pure propositional temporal logic (PTL), practically feasible algorithms exist for translating a PTL formula (which is usually short as a specification) into an equivalent Büchi automaton.

As Büchi automata are also suitable as abstract system models, many researchers have advocated a unified model-checking approach based on automata. In this approach, the negation of the specification formula is translated into an automaton, representing the bad behaviors. The intersection of the system automaton and the negated-specification automaton is then constructed and checked for emptiness. If the intersection automaton accepts no input (i.e., the system and the negated specification do not have any common behavior), then the system is correct with respect to the original specification formula.

Translation of a PTL formula into an equivalent Büchi automaton is now routinely performed in many model-checking algorithms and tools. Albeit 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, while Büchi automata "localize" temporal dependency to relations between states and tend to be of lower level. Nonetheless, their relation 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, however, is tedious and prone to mistakes for the students, while preparing the material is very time-consuming for the instructor. Tool support is needed.

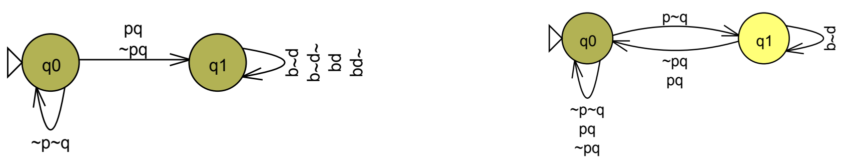
In this paper, we introduce a graphical interactive tool, named GOAL (**G**raphical Interactive Tool for **O**mega-**A**utomata and Temporal **L**ogic), that has been designed and implemented for this purpose, and suggest possible usages and benefits of the tool in courses where model-checking techniques are covered. GOAL builds on the successful JFLAP tool for classic theory of automata and formal languages. One main function of GOAL is translation of a PTL formula into an equivalent Büchi automaton that can be visually manipulated, for example, running the automaton on some input. The user has an option of viewing the intermediate steps that the translation goes through, in particular, which states of the automaton come from which parts of the input temporal formula. GOAL also supports various standard operations and tests, including equivalence test, on Büchi automata. We believe that, with an easy access to temporal formulae and their graphically presented equivalent Büchi automata, the student's understanding of the two formalisms and their relation will be greatly enhanced.

To the best of our knowledge, GOAL is the first graphical interactive tool designed mainly for teaching and learning Büchi automata and linear temporal logic. It supports the full range of temporal operators, including all past and future temporal operators defined in Manna and Pnueli's book [11]. There are other tools that provide translation of PTL formulae into Büchi automata, e.g., SPIN [8] and LTL2BA [5]. However, none of them provide facilities for visually manipulating automata and few support past temporal operators. The operations and tests on Büchi automata provided by GOAL are also more comprehensive than those by other tools.

2 Büchi Automata, Linear Temporal Logic, and Model Checking

Büchi Automata. Büchi automata are a variant of ω-automata, namely finite automata operating on infinite words. A Büchi automaton accepts those inputs that can drive it through some accepting state infinitely many times. Büchi automata are closed under intersection and complementation [1,7]. Complementation, unlike in the case of finite words, is highly complex [16,14,9,10,4] and known to have an exponential lower bound [12]. Minimizing the number of states is also a hard problem [3,17]. Generalized Büchi automata have multiple sets of accepting states. A generalized Büchi automaton accepts those inputs that can drive it through each of the accepting sets infinitely many times. Generalized Büchi automata and many other variants of ω-automata are equivalent to Büchi automata in expressive power.

Linear Temporal Logic. Linear temporal logic (LTL) has as semantic models infinite sequences of states, which can also be seen as infinite words over a suitable alphabet. We use Propositional Temporal Logic (PTL) to refer to the pure propositional version of LTL, for which a state is simply a subset of atomic propositions holding in that state. PTL formulae are constructed by applying boolean connectives and temporal operators to atomic propositions drawn from a predefined universe. For instance, the formula $\Box(p \rightarrow \Diamond q)$ combines two temporal operators, $\Box$ (always) and $\Diamond$ (once), to say that "every p is preceded by q" or equivalently "the first p does not occur before the first q". The formula $\Box(p \rightarrow p\ \mathscr{U}\ q)$ says that "once p becomes true, it will remain true continuously until q becomes true, which must eventually occur".



(a) $\Box(p \rightarrow \Diamond q)$ (b) $\Box(p \rightarrow p\ \mathscr{U}\ q)$

Fig. 1. Two PTL formulae and their respective equivalent Büchi automata, where the darker states are accepting states.

A PTL formula can be translated into an equivalent Büchi automaton (but not vice versa) in the sense that every infinite sequence satisfying the formula corresponds to an infinite word accepted by the automaton [19,5,6]. As an illustration, we exam the

Büchi automata equivalent to the two example temporal formulae. The alphabet for both automata is $\{$`pq`, `p~q`, `~pq`, `~p~q`$\}$. The Büchi automaton in Figure 1(a) is equivalent to the formula $\Box(p \rightarrow \Diamond q)$. From state q_0, there is no transition for `p~q`, ensuring that "the first p does not occur before the first q". The Büchi automaton in Figure 1(b) is equivalent to the formula $\Box(p \rightarrow p\ \mathcal{U}\ q)$. An occurrence of `p~q` brings the automaton from q_0 to q_1, where no transition is possible for `~p~q`. So, once p becomes true, it has to remain true until q becomes true. In addition, as q_1 is not an accepting state, either `pq` or `~pq` must occur, bringing the automaton to the accepting state q_0.

Another variant of LTL called Quantified Propositional Temporal Logic (QPTL) [15] additionally allows quantification over atomic propositions. QPTL are equivalent to Büchi automata in expressive power, though the translation from formula to automaton involves a non-elementary blow-up of number of states [16].

Model Checking. Model checking seeks to automatically verify if a given system satisfies its specification [2]. The system is typically modeled as a Kripke structure, which is essentially a state-transition graph where each state is labeled with those propositions that hold in that state; fairness may be imposed on how the transitions should be taken. When the specification is given by a PTL formula, the model checker determines if every computation (sequence of states) generated by the system satisfies (or is a model of) the formula.

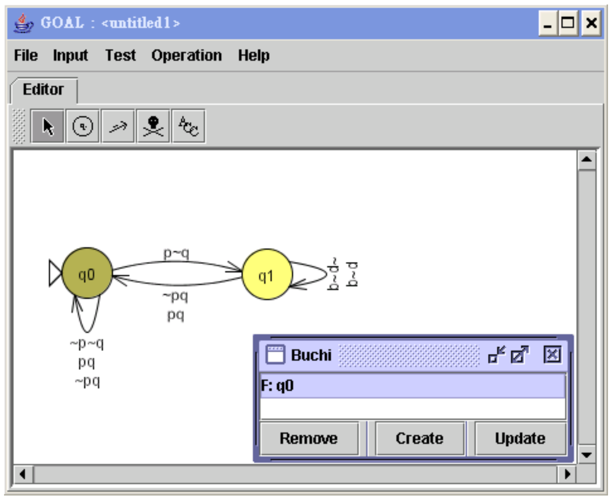
The system may also be modeled as a Büchi automaton; in fact, every Kripke structure (with or without the usual fairness conditions) corresponds to some Büchi automaton. As the PTL formula can also be translated into a Büchi automaton, this results in a uniform treatment of both the system and its specification [18]. Suppose A is the automaton modeling the system and B_φ the automaton representing the specification φ. Let $L(A)$ and $L(B_\varphi)$ denote respectively the languages of the two automata. The problem of model checking translates into that of language containment $L(A) \subseteq L(B_\varphi)$. Let $\overline{L(B_\varphi)}$ denote the complement of $L(B_\varphi)$. The problem is then equivalent to checking if $L(A) \cap \overline{L(B_\varphi)} = \emptyset$. As Büchi automata are closed under intersection and complementation, this reduces to the emptiness problem of Büchi automata.

However, complementation of a Büchi automaton is expensive. A better alternative is to first negate the given PTL formula φ and obtain the equivalent automaton $B_{\neg\varphi}$ such that $L(B_{\neg\varphi}) = \overline{L(B_\varphi)}$. Now, to check if $L(A) \cap L(B_{\neg\varphi}) = \emptyset$, one only needs to construct the intersection of A and $B_{\neg\varphi}$ and complementation is avoided.

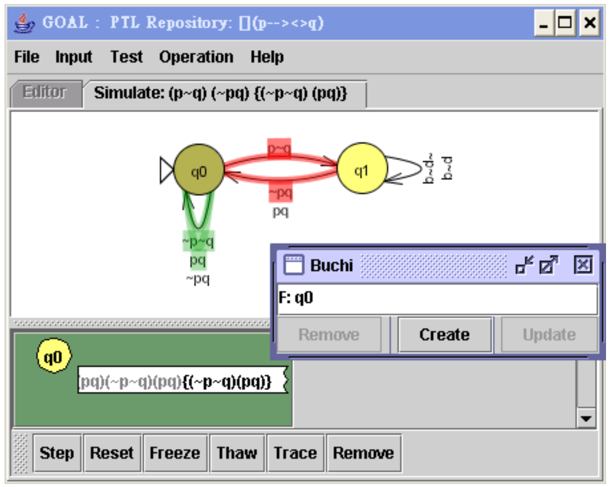
3 The GOAL Tool

In this section, we describe the functions of GOAL (**G**raphical Interactive Tool for **O**mega-**A**utomata and Temporal **L**ogic) and their implementation. The current version of GOAL provides the following functions:

- **Drawing and Running Büchi Automata**: The user can easily point-and-click and drag-and-drop to create a Büchi automaton; the automata in Figure 1 were drawn using GOAL. After an automaton is created, the user can run it through some input to get a feel of what kind of inputs the automaton accepts, as shown in Figure 2.

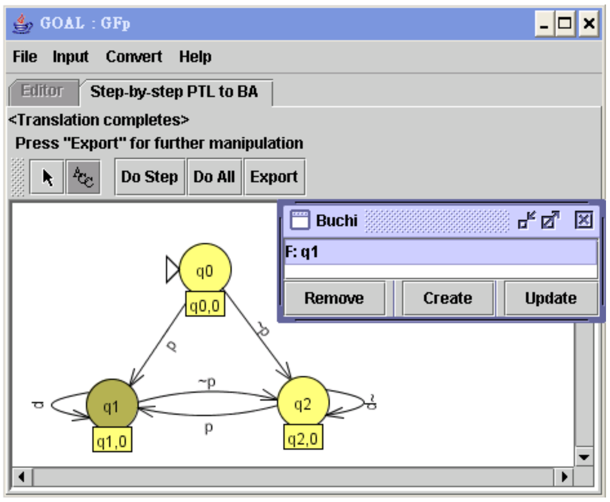


(a) A Büchi automaton drawn by the user

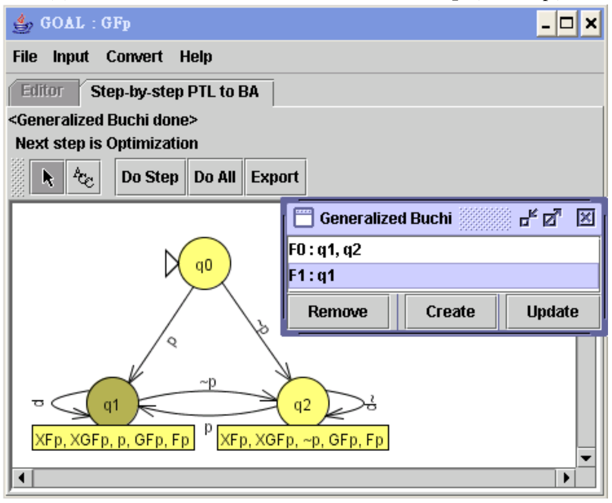

GOAL : PTL Repository: [](p-->&lt;&gt;q)
File Input Test Operation Help
Editor
Simulate: (p~q) (~pq) {(~p~q) (pq)}
q0
q1
p~q
~pq
pq
~p~q
pq
~pq
Buchi
F: q0
Remove Create Update
q0
pq)(~p~q)(pq){(~p~q)(pq)}
Step Reset Freeze Thaw Trace Remove

(b) Running the Büchi automaton through an input

Fig. 2. Screen shots of the GOAL tool. The inset window in each screen shot shows the set of accepting states. In Part (b), the pair of "{" and "}" in the input indicates an infinite repetition.



(a) A Büchi automaton translated from `GFp` (or $\Box\Diamond p$)



(b) The intermediate generalized Büchi automaton

Fig. 3. More screen shots of the GOAL tool. The PTL formula `GFp` is translated into an equivalent Büchi Automaton. If the translation is followed step by step, the user can also see the intermediate generalized Büchi automaton.

- **From PTL Formulae to Büchi Automata**: After the user has typed in a PTL formula and chosen a suitable translation option, the system responds by displaying an equivalent Büchi automaton, as shown in Figure 3(a) with "`GFp`" or equivalently "$\Box\Diamond p$" as input. With another option, the system first displays a generalized Büchi automaton and then, upon the user's request, will convert it into a Büchi automaton; Figure 3(b) shows such an intermediate result. The supported temporal operators and their input formats are as follows:

Operator	$\bigcirc$	$\Box$	$\Diamond$	$\mathscr{U}$	$\mathscr{W}$	$\mathscr{R}$	$\ominus$	$\circleddash\!\!\!\sim$	$\boxminus$	$\Diamond\!\!\!-$	$\mathscr{S}$	$\mathscr{B}$
Format 1	`()`	`[]`	`<>`	`U`	`W`	`R`	`(-)`	`(~)`	`[-]`	`<->`	`S`	`B`
Format 2	`X`	`G`	`F`	`U`	`W`	`R`	`Y`	`Z`	`H`	`O`	`S`	`B`

- **The PTL Repository**: The repository stores a collection of commonly seen patterns of PTL formulae and their respective equivalent Büchi automata, which were drawn by human using GOAL itself and are smaller than machine-translated ones.
- **Boolean Operations on Büchi Automata**: The three standard boolean operations—union, intersection, and complementation are supported.
- **Tests on Büchi Automata**: Emptiness, (language) containment, and equivalence tests are supported. In the emptiness test, if the given Büchi Automaton is nonempty, the system highlights the path that corresponds to an accepted input. The equivalence test of two Büchi Automata is built on top of the containment test which in turns relies on the intersection and complementation operations and the emptiness test.

The automata and the graph modules of GOAL were adapted from those of JFLAP [13]. The most complicated algorithms in GOAL are those for translating temporal formulae to automata and for complementing automata. Our translation algorithm is an adaptation of the tableau construction described in Chapter 5 of Manna and Pnueli's book [11]. For automata complementation, we adopted the algorithm by Safra [14]. Even with 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 very serious problem. As GOAL is positioned mainly as an instructional tool, we assume that it will be used with inputs of short temporal formulae or small automata.

A few words are in order about our translation algorithm. Though it generates more states than others do, the algorithm has two advantages: it handles past temporal operators and is relatively simple (which is good for educational purposes). The steps can be easily divided and their intentions clearly illustrated. Some published translation algorithms are indirect, e.g., the translation in [5] used a very weak alternating automaton as the intermediary, while some combine multiple steps into one, e.g., the translation in [6] constructed states and established transitions in the same step. To reduce the number of states, we implemented several methods for state reduction, for example, removing redundant states detected by simulation [3].

4 GOAL in Classroom and More

As the implementation of its main functions has just recently been completed, we have yet to use the GOAL tool in an actual classroom setting. However, an analogy can be drawn from the use of JFLAP [13], a visual interactive tool for teaching and learning classic theory of automata and formal languages that inspired GOAL. The first author has used JFLAP for several years in his Theory of Computation course for junior and senior undergraduate students. Both the students and the instructor have enjoyed the illuminating visualization that the tool provides. It helps to be able to *see* how an automaton, particularly a *nondeterministic* one, runs on an input. A convenient tool for drawing automata also encourages the students to do more exercises. It was a delight to find that a visual tool has breathed life into an important foundational computer science course that would otherwise be dull to most students. Moreover, as the diagrams can be exported with a PDF printing support, the tool has also saved the instructor's time when preparing handout material and the students' when writing up their homework. We believe the same will apply for GOAL.

The GOAL tool should be useful as teaching and learning support for courses on model checking, formal verification, or even advanced automata theory where ω-automata and temporal logic are essential topics. Our immediate plan is to use GOAL in a course of this Fall titled Software Development Methods, which aims at improving students' ability in designing quality software. The course covers three topics: software modeling, design patterns, and formal verification. In the verification part, we will cover model checking techniques and tools where GOAL can help. Though the emphasis is not on translation algorithms, the student will be asked to write the same specifications with Büchi automata and temporal formulae. With the help of GOAL (particularly the equivalence test), they will be able to quickly validate their answers. They can try out a few inputs on a Büchi automaton to get a better understanding of what its language is. For the more aspiring students, GOAL can provide them with guidance on how a Büchi automaton is obtained systematically from a PTL formula (though not necessarily in an optimal way).

Lastly, we would like to mention that the authors and other members in their group have already started to benefit from developing and using GOAL. Among other things, we learned to appreciate the difficulty and importance of minimizing the number of states of a Büchi automaton during the implementation. No known algorithms guarantee minimality on the number of states. Still any heuristics that help reduce the number of states are valuable, as automata with a smaller number of states are usually easier to understand intuitively (and to verify automatically). Indeed, research and teaching do go hand in hand.

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. Without the automata and the graph modules of JFLAP (which we had to tweak a little bit), the GOAL tool would not be possible in a relatively short time. We also thank Wen-Chin Chan who helped us with the adaptation of JFLAP's graph module.

References

1. J.R. Büchi. On a decision method in restricted second-order arithmetic. In *Proceedings of the International Congress on Logic, Methodology and Philosophy of Science*, pages 1–11. Standford University Press, 1962.

2. E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking*. The MIT Press, 1999.
3. K. Etessami and G. Holzmann. Optimizing Büchi automata. In *CONCUR 2000, LNCS 1877*, pages 153–167. Springer, 2000.
4. E. Friedgut, O. Kupferman, and M.Y. Vardi. Büchi complementation made tighter. In *Proceedings of the 2nd International Symposium on Automated Technology for Verification and Analysis, LNCS 3299*, pages 64–78. Springer, 2004.
5. P. Gastin and D. Oddoux. Fast LTL to Büchi automata translations. In *Proceedings of the 13th International Conference on Computer-Aided Verification, LNCS 2102*, pages 53–65. Springer, 2001.
6. 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.
7. E. Grädel, W. Thomas, and T. Wilke. *Automata, Logics, and Infinite Games (LNCS 2500)*. Springer, 2002.
8. G.J. Holzmann. *The SPIN Model Checker: Primer and Reference Manual*. Addison-Wesley, 2003.
9. N. Klarlund. Progress measures for complementation of ω-automata with application to temporal logic. In *Proceedings of the 32nd IEEE Conference on Foundations of Computer Science*, pages 358–367, 1991.
10. O. Kupferman and M. Vardi. Weak alternating automata are not that weak. *ACM Transactions on Computational Logic*, 2(3):408–429, 2001.
11. Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safty*. Springer, 1995.
12. M. Michel. Complementation is more difficult with automata on infinite words. In *CNET, Paris*, 1988.
13. S. Rodger and T. Finley. JFLAP. http://www.jflap.org/.
14. S. Safra. On the complexity of ω-automta. In *Proceedings of the 29th IEEE Conference on Foundations of Computer Science*, pages 319–327, 1988.
15. A.P. Sistla. *Theoretical Issues in the Design and Verification of Distributed Systems*. PhD thesis, Harvard University, 1983.
16. 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.
17. F. Somenzi and R. Bloem. Efficient Büchi automata from LTL formulae. In *Proceedings of the 12th International Conference on Computer-Aided Verification, LNCS 1855*, pages 248–263. Springer, 2000.
18. M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proceedings of the 1st IEEE Symposium on Logic in Computer Science*, pages 332–344, 1986.
19. P. Wolper. The tableau method for temporal logic: An overview. *Logique et Analyse*, 28(110):119–136, 1985.