Under consideration for publication in Formal Aspects of Computing

Tool Support for Learning Büchi Automata and Linear Temporal Logic[1]

Yih-Kuen Tsay, Yu-Fang Chen, Ming-Hsien Tsai, Kang-Nien Wu, Wen-Chin Chan, Chi-Jian Luo, and Jinn-Shu Chang
Department of Information Management, National Taiwan University, Taiwan

Abstract. We introduce a graphical interactive tool, named GOAL, that can assist the user in understanding Büchi automata, linear temporal logic, and their relation. Büchi automata and linear temporal logic are closely related and have long served as fundamental building blocks of linear-time model checking. Understanding their relation is instrumental in discovering algorithmic solutions to model checking problems or simply in using those solutions, e.g., specifying a temporal property directly by an automaton rather than a temporal formula so that the property can be verified by an algorithm that operates on automata.

One main function of the GOAL tool is translation of a temporal formula into an equivalent Büchi automaton that can be further manipulated visually. The user may edit the resulting automaton, attempting to optimize it, or simply run the automaton on some inputs to get a basic understanding of how it operates. GOAL includes a large number of translation algorithms, most of which support past temporal operators. With the option of viewing the intermediate steps of a translation, the user can quickly grasp how a translation algorithm works. The tool also provides various standard operations and tests on Büchi automata, in particular the equivalence test which is essential for checking if a hand-drawn automaton is correct in the sense that it is equivalent to some intended temporal formula or reference automaton. Several use cases are elaborated to show how these GOAL functions may be combined to facilitate the learning and teaching of Büchi automata and linear temporal logic.

Keywords: Büchi Automata; GOAL; Linear Temporal Logic; Model Checking; QPTL

---

[1] This work was partially supported by the National Science Council, Taiwan (R.O.C.) under grants NSC94-2213-E-002-089, NSC95-2221-E-002-127, NSC95-3114-P-001-001-Y02 (iCAST 2006), NSC96-3114-P-001-002-Y (iCAST 2007), and NSC97-2221-E-002-074-MY3.

Correspondence and offprint requests to: Yih-Kuen Tsay, Dept. of Information Management, National Taiwan University, No. 1, Sec. 4, Roosevelt Rd., Taipei 106, Taiwan. Email: tsay@im.ntu.edu.tw

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 [CGP99]. 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 [Sha00].

In the so-called linear-time 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 of languages for specification. Temporal dependency between events may also be expressed with *Büchi automata*, which are finite automata operating on infinite words (that correspond to infinite computations). Indeed, Büchi automata and linear temporal logic are closely related. It has been shown that Büchi automata and a variant of linear temporal logic called quantified propositional temporal logic (QPTL) are expressively equivalent, though translation between the two formalisms is highly complex [KP02]. For the pure propositional temporal logic (PTL), practically feasible algorithms exist for translating a PTL formula into an equivalent Büchi automaton [KMMP93, GPVW95, DGV99, GO01], though not vice versa.

As Büchi automata are also suitable as abstract system models, many researchers have advocated a unified model-checking approach based on automata [VW86]. In this automata-theoretic approach, the negation of the temporal 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.

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, while Büchi automata "localize" temporal dependency to relations between states and tend to be of lower level. Understanding their relation is instrumental in discovering algorithmic solutions to model checking problems or simply in using those solutions, e.g., specifying a temporal property directly by an automaton rather than a temporal formula so that the property can be verified by an algorithm that operates on automata. To enhance this understanding, it helps to go through several translation algorithms 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 student, 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 (which stands for "**G**raphical Tool for **O**mega-**A**utomata and **L**ogics" and is available at `http://goal.im.ntu.edu.tw`), that has been designed and implemented for this purpose. One main function of the GOAL tool is translation of a QPTL formula into an equivalent Büchi automaton that can be further manipulated visually. The user may edit the resulting automaton, attempting to optimize it, or simply run the automaton on some inputs to get a basic understanding of how it operates. GOAL includes a large number of translation algorithms, most of which support past temporal operators. With the option of viewing the intermediate steps of a translation, the user can quickly grasp how a translation algorithm works. The tool also provides various standard operations and tests on Büchi automata, in particular the equivalence test which is essential for checking if a hand-drawn automaton is correct in the sense that it is equivalent to some intended temporal formula or reference automaton. Several use cases are elaborated to show how these GOAL functions may be combined to facilitate the learning and teaching of Büchi automata and linear temporal logic. 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 for learning and teaching Büchi automata and linear temporal logic. It supports past temporal operators and quantification over propositional variables. There are other tools that provide translation of temporal formulae into Büchi automata, e.g., SPIN [Hol03], LTL2BA [GO01], Wring [SB00], MoDeLLa [ST03], and LTL2Buchi [GL02]. SPIN in particular is an automata-theoretic model checker that has been widely used both in practice and in education. None of these tools 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.

Earlier versions of GOAL have been introduced and suggested for educational purposes in an informal workshop [TCW06] and for supplementing automata-theoretic model checkers such as SPIN in a conference [TCT+07]. Compared to these earlier versions, the version of GOAL described here includes a much larger collection of translation, simplification, and complementation algorithms. This should meet the needs of more users. Moreover, an option to play out the intermediate steps of a translation is provided for most of the translation algorithms. This should expedite the learning of a translation algorithm and hence the understanding of the relation between a temporal formula and its equivalent Büchi automaton. The LTL2BA tool can also show intermediate steps; however, this is done in texts and is not very friendly for the learner. More recently, we have also started to explore the usages of GOAL as a research tool [TCT+08].

The rest of this paper is organized as follows. Section 2 gives a brief overview of Büchi automata, linear temporal logic, and their roles in model checking. In Section 3, we present the GOAL tool, detailing its main functions along with some highlights of their implementation. In Section 4, several basic usages of GOAL are elaborated for educational purposes. Three more advanced examples can be found in Section 5. Section 6 concludes with some remarks.

2. Büchi Automata, Linear Temporal Logic, and Model Checking

This section gives a brief overview of Büchi automata and linear temporal logic along with their roles in model checking. The reader who is familiar with these subjects may safely skip this section. For the reader who is not familiar with these subjects and wishes to know more about them, a more detailed tutorial with precise formal definitions can be found in the Appendix.

Büchi Automata Büchi automata are a variant of ω-automata, which are finite-state automata operating on infinite words. A Büchi automaton accepts those inputs that can drive it through some accepting state infinitely many times. Two examples of Büchi automata will be given subsequently when we contrast them with their equivalent temporal formulae. (Nondeterministic) Büchi automata are closed under intersection and complementation [Büc62, GTW02]. Complementation of a Büchi automaton, unlike in the case of finite words, is a hard problem and has a well-known exponential worst-case lower bound of $2^{\Omega(n \log n)}$ [Mic88]. Solutions to this problem are often complicated and difficult to learn [SVW87, Saf88, Kla91, KV01, FKV04, Pit06]. Minimizing the number of states of a Büchi automaton is also a hard problem [EH00, SB00].

Generalized Büchi automata have multiple sets of accepting states. They naturally arise as intermediate forms in the translation of temporal formulae into Büchi automata. A generalized Büchi automaton accepts those inputs that can drive it through some state of each accepting set 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 its 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* (“not” $\neg$, “or” $\vee$, “and” $\wedge$, “implies” $\rightarrow$, “is equivalent to” $\leftrightarrow$) and *temporal* (“next” $\bigcirc$, “eventually” $\Diamond$, “always” $\Box$, “until” $\mathcal{U}$, “wait-for” $\mathcal{W}$, “previous” $\ominus$, “before” $\circleddash$, “once” $\diamondminus$, “so-far” $\boxminus$, “since” $\mathcal{S}$, “back-to” $\mathcal{B}$) operators to atomic propositions drawn from a predefined universe. For instance, the formula $\Box(p \rightarrow \diamondminus q)$ is obtained by applying $\rightarrow$, $\Box$, and $\diamondminus$ to atomic propositions p and q, saying that “every p is preceded by a q” or equivalently “the first p does not occur before the first q”. The formula $\Box(p \rightarrow p\,\mathcal{U}\,q)$ says that “once p becomes true, it will remains true continuously until q becomes true, which must eventually occur”. In the literature, there exist two versions of PTL. One contains both past and future temporal operators (for example, in Manna and Pnueli’s books [MP92, MP95]), while the other contains only future operators (for example, in Clarke *et al.* [CGP99], referred to as LTL there). Although these two versions are equivalent in expressive power, past operators provide a more concise and intuitive way for constructing some specifications [LPZ85].

Every PTL formula can be translated into an equivalent Büchi automaton (but not vice versa) in the sense that each infinite sequence satisfying the formula corresponds to an infinite word accepted by the automaton [KMMP93, GO03]. As an illustration, we examine the Büchi automata that are equivalent to

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

Fig. 1. Two PTL formulae and their respective equivalent Büchi automata. In the state transition diagrams, True is a shorthand representing any input symbol from the alphabet set {pq, p∼q, ∼pq, ∼p∼q}, ∼p any input symbol from {∼pq, ∼p∼q}, and q any input symbol from {pq, ∼pq}. The double-circled states in darker shade are accepting states.

the two example PTL formulae described in the previous paragraph. The alphabet for both automata is {pq, p∼q, ∼pq, ∼p∼q} (a set of four "structured" symbols). The Büchi automaton in Figure 1(a) is equivalent to the formula $\Box(p \rightarrow \Diamond q)$. From the initial state s_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 s_0 to s_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 s_1 is not an accepting state, either pq or ∼pq must occur, bringing the automaton to the accepting state s_0.

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), referred to as "Even p" here, is a typical example for showing the difference. (In [Wol83], Wolper proved that, for any given $m \geq 2$, the property "p is true at every (km)-th position, where $k \geq 0$" cannot be specified by PTL.) Quantified Propositional Temporal Logic (QPTL) [Sis83] extends PTL by additionally allowing quantification over atomic propositions. With the extension, QPTL is equivalent to Büchi automata in expressive power. Every QPTL formula can be algorithmically translated into an equivalent Büchi automaton and vice versa [KP02].

Model Checking Model checking seeks to automatically verify if a given system satisfies its specification [CGP99]. The system is typically modeled as a Kripke structure—a state-transition graph where each state is labeled with those propositions that hold in the state; fairness may be imposed on how often the states should be visited. When the specification is given by a linear temporal logic formula, the model checking problem is to determine if every computation (sequence of states) generated by the Kripke structure satisfies, or is a model of, the temporal formula.

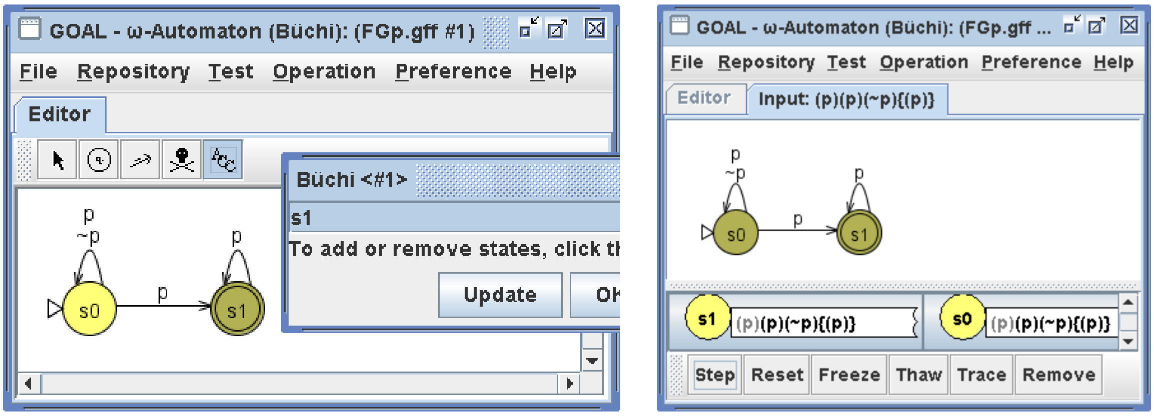
The system may also be modeled as a Büchi automaton; in fact, every Kripke structure (with or without the usual fairness conditions) corresponds to some Büchi automaton. As the specification formula can also be translated into a Büchi automaton, this results in a uniform treatment of both the system and its specification [VW86]. 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 $L(B_\varphi)$ and $\overline{B_\varphi}$ the complement of B_φ. The problem is then equivalent to checking if $L(A) \cap \overline{L(B_\varphi)} = \emptyset$, i.e., $L(A \cap \overline{B_\varphi}) = \emptyset$. As Büchi automata are closed under complementation and intersection, this reduces to the emptiness problem of Büchi automata.

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

3. Functions of GOAL

In this section, we describe the main functions of GOAL along with some highlights of 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 or a generalized Büchi automaton; see Figure 2(a). After an automaton



(a) A Büchi automaton intended for $\Diamond\Box p$ (b) Running the automaton on $\mathsf{pp{\sim}pppp}\cdots$

Fig. 2. An example of drawing and running a Büchi automaton with GOAL (in a preference setting with slightly larger fonts for readability). The inset window in Part (a) is the dialog window for updating the accepting states. In Part (b), the pair of "{" and "}" in the input indicates an infinite repetition. The bottom pane shows every reachable configuration (including the state and the remaining input) of the automaton on the input, each box representing a configuration. In each box, the lighter part of the input has been consumed by the automaton (which leads the automaton to the state in the same box) and the darker part is the remaining input to be consumed. Currently, the first input symbol p has been consumed and the automaton is in either s0 or s1. When an accepting run is found, the path corresponding to the run will be highlighted.

Table 1. Major algorithms in GOAL. The Modified Safra algorithm for complementation is a slight variation of Safra's construction.

Translation
Tableau [MP95], Incremental Tableau [KMMP93], Temporal Tester [KP00], GPVW [GPVW95], GPVW+ [GPVW95], LTL2AUT [DGV99], LTL2AUT+ [TCTL07], LTL2BA [GO01], PLTL2BA [GO03]
Complementation
Kurshan [Kur87], Safra [Saf88], Modified Safra, Muller-Schupp [MS95], WAPA [Tho99], WAA [KVW00], Piterman [Pit06]
Simplification
Simulation [SB00], Pruning fair sets [SB00]

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(b).

- **Testing Büchi Automata**: Emptiness, universality, simulation relation, (language) containment, and equivalence tests are supported. In the emptiness test, if the given Büchi automaton is non-empty, GOAL highlights the path that corresponds to an accepted input. The equivalence test on two Büchi automata is built on top of the containment test which in turn relies on the intersection and complementation operations and the emptiness test. An equivalence test can also be performed between a Büchi automaton and a temporal formula.
- **Translating QPTL (and PTL) Formulae into Büchi Automata**: Nine algorithms have been implemented for temporal formula to Büchi automaton translation; see Table 1. It is also possible to translate a formula into a generalized Büchi automaton, instead of going all the way to a Büchi automaton. Four (Tableau, Incremental Tableau, Temporal Tester, and PLTL2BA) of them originally supported past operators. We have extended three more (GPVW, LTL2AUT, LTL2AUT+) to allow past operators. All these nine algorithms are further extended to support quantification on propositions. 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 unrestricted QPTL [SVW87]. The supported boolean and temporal operators and their input formats are shown in Table 2.
  To help learning, translations by five of the nine algorithms (Tableau, GPVW, GPVW+, LTL2AUT,

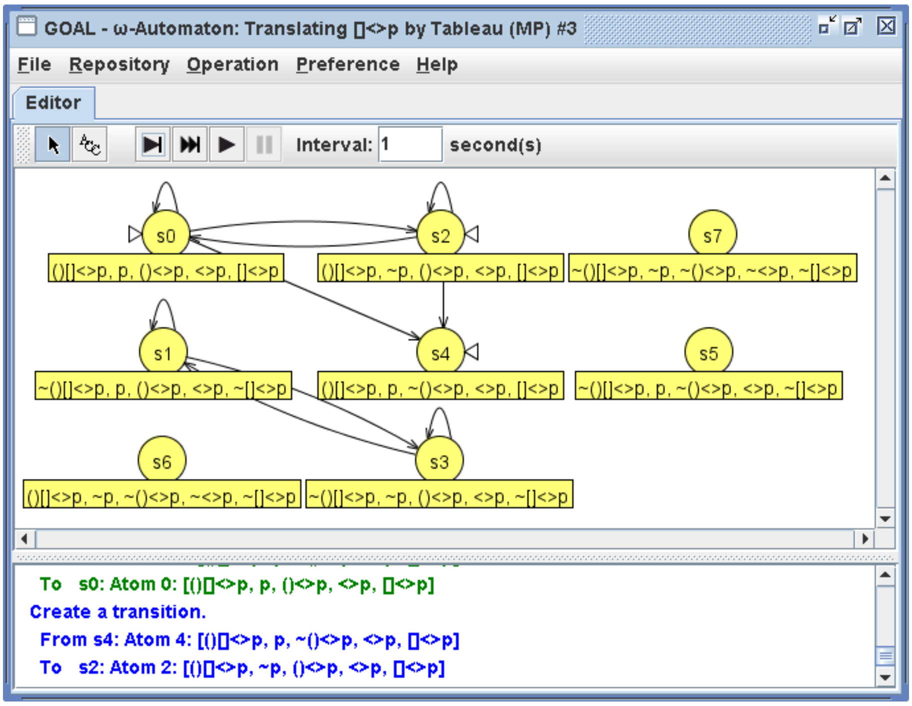


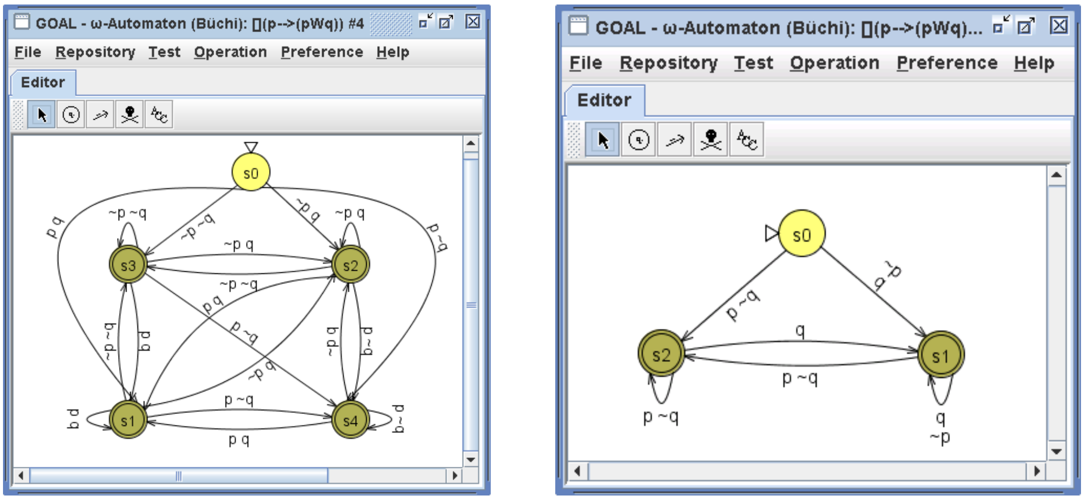
Fig. 3. A screen shot of the step-by-step translation of a temporal formula into an equivalent Büchi automaton using the Tableau algorithm. The given PTL formula is $\square\diamond p$. The lower window displays explanatory descriptions, while the steps are played out in the upper window.

Table 2. Boolean and temporal operators supported in GOAL and their input formats.

Operator	$\neg$	$\vee$	$\wedge$	$\rightarrow$	$\leftrightarrow$	$\bigcirc$	$\square$	$\diamond$	$\mathcal{U}$	$\mathcal{W}$	$\ominus$	$\widetilde{\ominus}$	$\boxminus$	$\diamondminus$	$\mathcal{S}$	$\mathcal{B}$	$\exists$	$\forall$
Format 1	~	\/	/\	-->	<-->	()	[]	<>	U	W	(-)	(~)	[-]	<->	S	B	E	A
Format 2	~	\/	/\	-->	<-->	X	G	F	U	W	Y	Z	H	O	S	B	E	A

LTL2AUT+) can be viewed step by step. For instance, in the Tableau algorithm the user gets to see the closure computed for the given temporal formula, atoms created as states of the automaton, and transitions between states added one by one. Text is displayed to explain the intermediate step being carried out. The user can "play" the translation, "pause" it, and then "resume" it. A snapshot of translating $\square\diamond p$ with the Tableau algorithm is shown in Figure 3.

- **Boolean Operations on Büchi Automata**: The three standard boolean operations—union, intersection, and complementation are supported. Büchi complementation is crucial in the implementation of language containment and equivalence tests, which are perhaps the most distinct functions of GOAL. Algorithms for Büchi complementation, because of their technical difficulty, are themselves a separate topic of learning (and also of research). Seven algorithms have been implemented in GOAL for Büchi complementation; see Table 1 for a listing. Cross-checking greatly increases our confidence in the correctness of the different complementation algorithms and hence the correctness of the language containment and equivalence tests.
  Complementation algorithms typically proceed in stages. For example, Safra's algorithm complements a Büchi automaton in three stages: (1) translate the given automaton into an equivalent deterministic Rabin automaton, (2) complement the Rabin automaton by interpreting it as a Streett automaton, and (3) translate the Streett automaton back into a Büchi automaton. (A formal definition of Rabin and Streett automata can be found in the Appendix.) The GOAL tool provides an option of showing the



(a) The automaton before simplification (b) The automaton after simplification

Fig. 4. A demonstration of the simplification algorithm.

results of intermediate stages for nearly all of the implemented complementation algorithms, which will be convenient for learning.

- **Tests on QPTL Formulae**: Satisfiability and validity tests are supported. The equivalence test between two formulae is not supported directly, but can be easily realized by connecting the two formulae with the mutual implication operator ($\leftrightarrow$) and testing the resulting formula for validity.

- **Simplifying Büchi Automata**: The user can use the simplification (by simulation) operation to find states of a Büchi automaton that simulate each other and merge those states; there is also an operation for simplifying generalized Büchi Automata by pruning fair sets (acceptance sets). The algorithm for finding simulation relations is an adaption of that proposed by Somenzi and Bloem [SB00]. Figure 4 shows an example of running the simplification algorithm on an automaton translated from the formula $\square(p \rightarrow p\ \mathcal{W}\, q)$ (once p becomes true, it will remain true continuously until q becomes true, which may never occur). To understand the original machine-translated automaton is somewhat difficult. After the simplification, one gets a smaller automaton, as shown in Figure 4(b), which is easier to understand.

- **Exporting Büchi Automata as Promela Code**: Once an automaton has been defined and tested, the user can export it in the Promela (the system modeling language of SPIN) syntax on the screen or as a file. This makes it possible to use GOAL as a graphical specification definition frontend to an automata-theoretic model checker like SPIN.

- **The Automata Repository**: This repository contains a collection of frequently used temporal formulae and their corresponding equivalent automata, which have been optimized by hand and checked by the GOAL tool itself. For beginners, this should be very convenient for learning the relation between Büchi automata and linear temporal logic.

GOAL is implemented in Java for the ease of installation. Its automaton and graph modules were adapted and extended from those of JFLAP [RF], which is a visual interactive tool for learning and teaching the classical theory of automata and formal languages. The most complicated algorithms implemented in GOAL are those for translating temporal formulae into automata and for complementing and simplifying automata, as summarized in Table 1. QPTL formula to Büchi automaton translation is done by combining one of the PTL formula to Büchi automaton translation algorithms with Sistla's approach for handling quantification [SVW87].

4. Basic Usages

We suggest in this section a number of basic use cases illustrating how the GOAL functions may be combined to facilitate the learning and teaching of Büchi automata and linear temporal logic. A few more advanced cases are discussed in the next section.

4.1. Viewing How a Büchi Automaton Operates on an Infinite Word

For a student who has taken a course on the classical theory of computation, the key to understanding Büchi automata is to first comprehend the concept of an infinite word and how a Büchi automaton operates on an infinite word. One obvious thing to do is examining a few examples of how an infinite word drives a Büchi automaton through the different states of the automaton, which can be conveniently carried out with GOAL.

A Büchi automaton for $\Diamond\Box p$ (p always holds from some point on) would be a simple enough starting example for illustrating how a Büchi automaton operates on infinite words. Suppose the alphabet is simply $\{\mathsf{p}, \sim\mathsf{p}\}$. In GOAL, an infinite word $\mathsf{pp}\sim\mathsf{pppp}\cdots$ (with p repeating indefinitely) is represented as $(\mathsf{p})(\mathsf{p})(\sim\mathsf{p})\{(\mathsf{p})\}$. Given this infinite word as input, the automaton for $\Diamond\Box p$ has infinitely many possible runs. The teacher can first explain the reason why the acceptance of an infinite word can be determined within a finite number of steps. GOAL can then be used to create the automaton, input the infinite word to the automaton, run the automaton for a few steps, find an accepting run, and explain again the reason why; Figure 2(b) shows a snapshot of one such scenario.

4.2. Translating a Temporal Formula into an Equivalent Büchi Automaton

Understanding how a temporal formula can be translated into a Büchi automaton is an essential step in learning automata-theoretic model checking. As we have explained earlier, temporal formulae and Büchi automata are very different artifacts and it can be difficult for the student to grasp their correspondence. In the translation function provided by GOAL, the user has an option of viewing the intermediate steps that a translation goes through. The visual aide can be very useful. For example, after studying a translation algorithm, the user can test his understanding of the algorithm by running the algorithm with paper and pencil and comparing each step with that generated by GOAL.

We suggest that beginners start with the tableau construction of Manna and Pnueli [MP95]. Though it generates more states than most others do, this algorithm is relatively simple and easy to understand. The steps can be easily divided and their intentions clearly described.

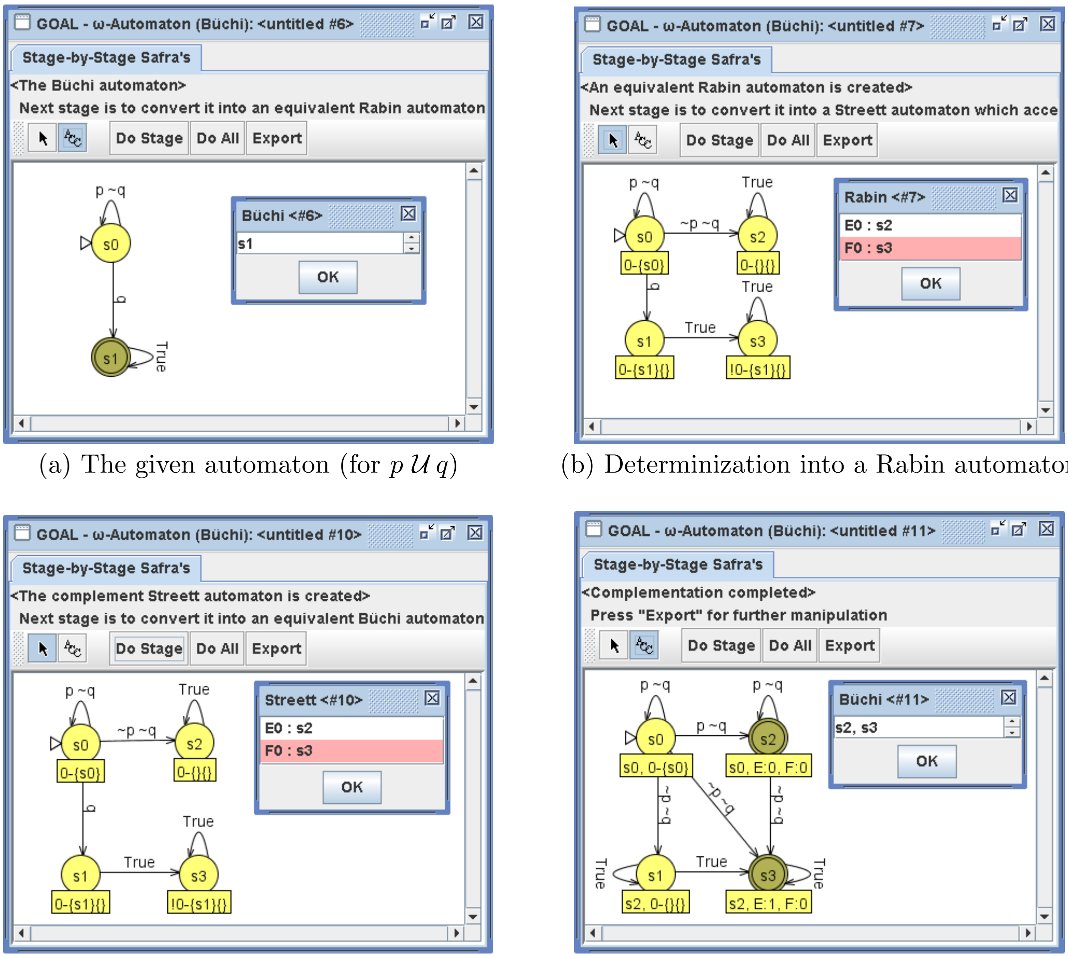
4.3. Performing Boolean Operations on Büchi Automata

Büchi automata are closed under boolean operations and these operations can be done algorithmically. To learn any of the boolean operations, the user can perform the operation by hand and then verify correctness by checking the equivalence between the resulting automaton (hand-drawn using the automaton editing function of GOAL) and the machine-computed one (also by GOAL).

GOAL is particularly useful for learning the complementation operation, which is very complex and difficult to understand. This again can be achieved by simulating an algorithm by hand and checking its correctness by machine. A stage-by-stage complementation with Safra's construction is shown in Figure 5.

4.4. Learning the Automata-Theoretic Model Checking Procedure

With the ability to translate temporal formulae into equivalent Büchi automata and perform boolean operations on Büchi automata, GOAL can be used for learning the basics of automata-theoretic model checking. It should be a helpful and interesting exercise for the student to go through the typical verification steps: (1) prepare a system Büchi automaton for some small verification problem, e.g., the two-process mutual exclusion problem, (2) write a temporal formula describing the system's safety property (e.g., mutual exclusion) or liveness property (e.g., starvation freedom), (3) negate the formula and translate it into a Büchi



(a) The given automaton (for $p\,\mathcal{U}\,q$)

(b) Determinization into a Rabin automaton

(c) Conversion into a Streett automaton

(d) Translation back into a Büchi automaton

Fig. 5. The stages in complementing a Büchi automaton by Safra's construction.

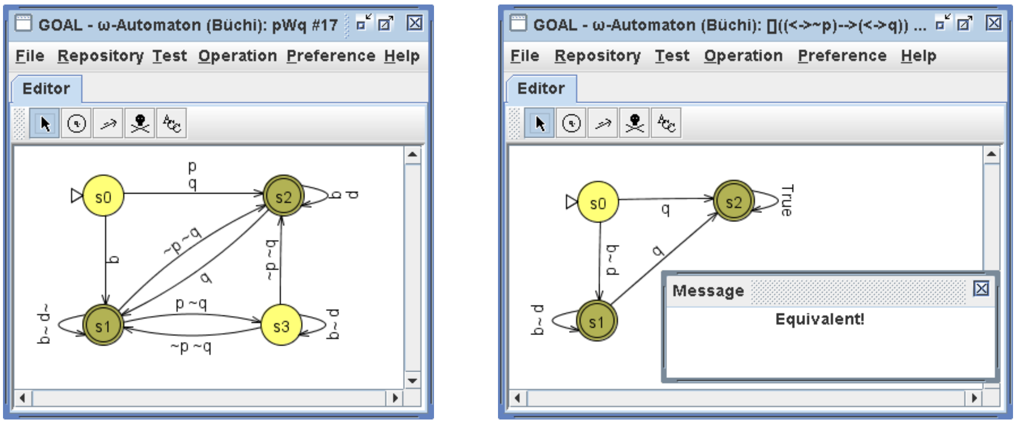
automaton, representing all "bad" behaviors, (4) compute the intersection of the given system automaton and the translated negative specification automaton, and (5) check the emptiness of the intersection.

4.5. Developing Specification Automata for a Model Checker

In SPIN, 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 specification automaton 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" (a Büchi automaton specifying all behaviors disallowed by the model) and continue the model checking procedure as usual.

5. Advanced Examples

Here we give three examples of using GOAL to help understand more difficult concepts in Büchi automata and linear temporal logic.



(a) The automaton generated from $p\,\mathcal{W}\,q$

(b) The automaton generated from $\Box(\diamondminus\neg p \rightarrow \diamondminus q)$

Fig. 6. A safety formula and its equivalent canonical formula. The inset window in Part (b) displays a message from the equivalence test, stating that the two automata are "Equivalent!".

5.1. Learning Safety Properties and Safety Formulae

Safety properties are requirements that should be met continuously by the system. A temporal formula is called a *safety* formula (specifying a safety property) if it is equivalent to some formula in the *canonical* form $\Box p$, where p is a past formula (which contains no future operators) [MP90, MP95]. The correspondence between a formula and its equivalent canonical safety formula can be hard to recognize. For example, the formula $p\,\mathcal{W}\,q$ (read "p wait-for q", which means p holds until an occurrence of q or p holds forever) is a safety formula, because it is equivalent to the canonical safety formula $\Box(\diamondminus\neg p \rightarrow \diamondminus q)$. The equivalence is not intuitive, but it can be easily verified with GOAL by either checking the validity of $p\,\mathcal{W}\,q \leftrightarrow \Box(\diamondminus\neg p \rightarrow \diamondminus q)$ or translating both formulae into Büchi automata and checking their equivalence, as shown in Figure 6. Further examples include $\Box p \vee \Box q \leftrightarrow \Box(\boxminus p \vee \boxminus q)$, $\neg(p\,\mathcal{U}\,\neg q) \leftrightarrow \Box(\ominus\boxminus p \rightarrow q)$, etc.

5.2. Understanding Why "Even p" Is QPTL-Expressible but Not PTL-Expressible

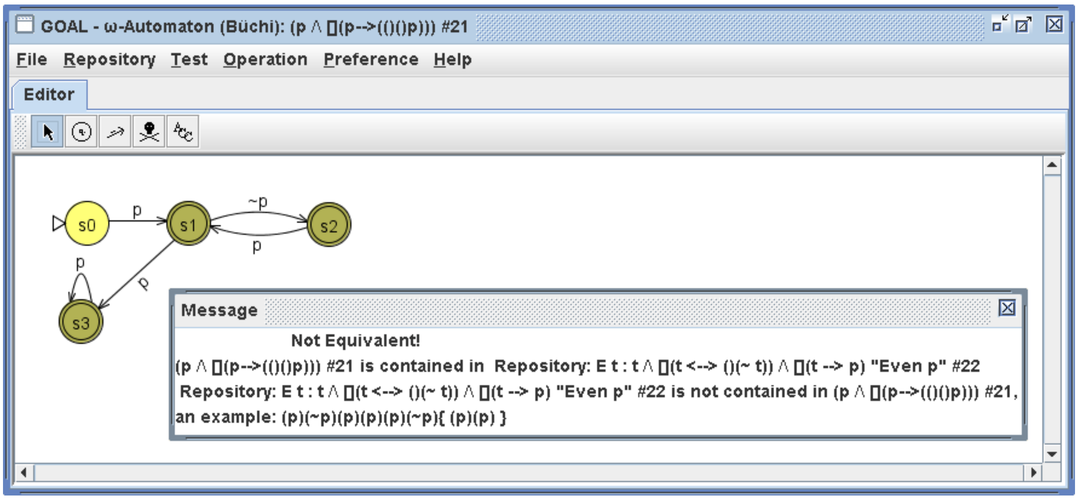
"Even p", as discussed in Section 2, is a typical case for showing PTL is strictly less expressive than Büchi automata. A plausible PTL formula for the property would be "$p \wedge \Box(p \rightarrow \bigcirc\bigcirc p)$". We translate the formula into a Büchi automaton, as shown in Figure 7(a), and open the "Even p" case in the repository, as shown in Figure 7(b). An equivalence test shows that the two automata are not equivalent and displays a counter-example, as shown in Figure 7(a) (again, an infinite word or sequence starts with position 0). 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 following odd positions, which is not required by "Even p".

A correct QPTL formula is $\exists t : t \wedge \Box(t \leftrightarrow \neg\bigcirc t) \wedge \Box(t \rightarrow p)$. In this formula, t is an auxiliary variable that is true at all even positions and false at all odd positions along a computation. From the subformula $\Box(t \rightarrow p)$, we know that p must be true when t is true (at even positions), but p can be any value when t is false (at odd positions). The formula is translated into a Büchi automaton, as shown in Figure 7(c). One can perform an equivalence test on the translated automaton and the one in the repository to be assured that the formula indeed expresses "Even p".

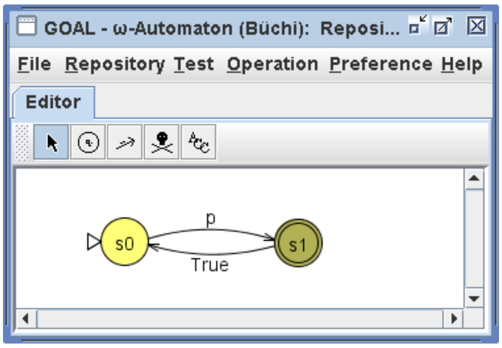
5.3. Understanding Temporal Assume-Guarantee Formulae

Informally, an assume-guarantee specification asserts that "some property is guaranteed while the assumption holds". In the literature [CGP99, Tsa00, NT00], we can find at least three temporal logic formulations:

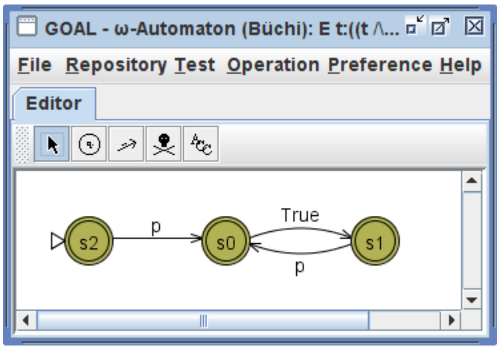
1. $\neg(p\,\mathcal{U}\,\neg q)$



(a) An automaton machine-translated from $p \wedge \Box(p \rightarrow \bigcirc\bigcirc p)$



(b) The "Even p" case from the repository



(c) An automaton machine-translated from $\exists t : t \wedge \Box(t \leftrightarrow \neg\bigcirc t) \wedge \Box(t \rightarrow p)$

Fig. 7. Automata intended for "Even p"; the one in (a) is incorrect.

2. $\Box(\ominus \boxminus p \rightarrow q)$
3. $q \, \mathcal{W} \, (\neg p \wedge q)$

Though quite different in appearance, all these three formulae are in fact equivalent, which can be easily confirmed with GOAL. There is another similar but weaker formula $\Box(\boxminus p \rightarrow \boxminus q)$ [Tsa00]. The formula can be translated into an equivalent Büchi automaton and checked to be *inequivalent* to any of the previous three formulae. Counter-examples from the tests should be helpful in understanding the difference.

6. Conclusion

We have described GOAL and suggested possible usages of the tool. To draw an analogy with JFLAP, we expect GOAL to be useful as learning and teaching support for courses on model checking, formal verification, or advanced automata theory where ω-automata and temporal logic are essential topics. It helps to be able to *see* how an automaton, particularly a *nondeterministic* one, runs on a given input. A convenient tool for drawing automata or generating automata from formulae also encourages the students to do more exercises and enhance their understanding of the subjects.

The first author of this paper has used GOAL in his "Software Development Methods" course, where linear-time model checking is covered. Although the emphasis is not on translation algorithms, the students were asked to write the same specifications with Büchi automata and temporal formulae. With the help of

GOAL, particularly the equivalence test, they were able to quickly validate their answers. They would also try out a Büchi automaton on several inputs to get a better understanding of what its language is. For the more aspiring students, GOAL provides them with guidance on how a Büchi automaton can be obtained systematically from a QPTL formula (though not necessarily in an optimal way).

As the source "**G**raphical Tool for **O**mega-**A**utomata and **L**ogics" of the acronym GOAL suggests, our long-term goal is for the tool to handle the common variants of ω-automata and the logics that are expressively equivalent to these automata. For example, besides Büchi and generalized Büchi automata, we have extended GOAL to support the editing of and a limited set of operations on Muller, Rabin, Streett, and Parity automata [GTW02]. Although these variants of ω-automata do not necessarily have a direct impact on the model-checking process, they are powerful intermediaries for the development of automata-based algorithms and will make GOAL complete as a learning and teaching tool.

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

References

[Büc62] J.R. Büchi. On a decision method in restricted second-order arithmetic. In *Proceedings of the 1960 International Congress on Logic, Methodology and Philosophy of Science*, pages 1–11. Standford University Press, 1962.

[CGP99] E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking*. The MIT Press, 1999.

[DGV99] M. Daniele, F. Giunchiglia, and M.Y. Vardi. Improved automata generation for linear temporal logic. In *Proceedings of the 11th International Conference on Computer-Aided Verification (CAV 1999), LNCS 1633*, pages 249–260. Springer, 1999.

[EH00] 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.

[FKV04] 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 (ATVA 2004), LNCS 3299*, pages 64–78. Springer, 2004.

[GL02] D. Giannakopoulou and F. Lerda. From states to transitions: Improving translation of LTL formulae to Büchi automata. In *Proceedings of the 22nd IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems (FORTE 2002), LNCS 2529*, pages 308–326. Springer, 2002.

[GO01] P. Gastin and D. Oddoux. Fast LTL to Büchi automata translations. In *Proceedings of the 13th International Conference on Computer-Aided Verification (CAV 2001), LNCS 2102*, pages 53–65. Springer, 2001.

[GO03] P. Gastin and D. Oddoux. LTL with past and two-way very-weak alternating automata. In *Proceedings of the 28th International Symposium on Mathematical Foundations of Computer Science (MFCS 2003), LNCS 2747*, pages 439–448. Springer, 2003.

[GPVW95] 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.

[GTW02] E. Grädel, W. Thomas, and T. Wilke. *Automata, Logics, and Infinite Games (LNCS 2500)*. Springer, 2002.

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

[Kla91] N. Klarlund. Progress measures for complementation of ω-automata with application to temporal logic. In *Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science (FOCS 1991)*, pages 358–367. IEEE Computer Society, 1991.

[KMMP93] Y. Kesten, Z. Manna, H. McGuire, and A. Pnueli. A decision algorithm for full propositional temporal logic. In *Proceedings of the 5th International Conference on Computer-Aided Verification (CAV 1993), LNCS 697*, pages 97–109. Springer, 1993.

[KP00] Y. Kesten and A. Pnueli. Verification by augmented finitary abstraction. *Information and Computation*, 163:203–243, 2000.

[KP02] Y. Kesten and A. Pnueli. Complete proof system for QPTL. *Journal of Logic and Computation*, 12(5):701–745, 2002.

[Kur87] R.P. Kurshan. Complementing deterministic Büchi automata in polynomial time. *Journal of Compututer and System Science*, 35:59V71, 1987.

[KV01] O. Kupferman and M.Y. Vardi. Weak alternating automata are not that weak. *ACM Transactions on Computational Logic*, 2(3):408–429, 2001.

[KVW00] O. Kupferman, M.Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *Journal of the ACM*, 47(2):312–360, 2000.

[LPZ85] O. Lichtenstein, A. Pnueli, and L. Zuck. The glory of the past. In *Proccedings of the Workshop on Logics of Programs, LNCS 193*, pages 196–218. Springer, 1985.

[Mic88] M. Michel. Complementation is more difficult with automata on infinite words. In *CNET, Paris*, 1988.

[MP90] Z. Manna and A. Pnueli. A hierarchy of temporal properties. In *Proceedings of the 9th ACM Symposium on Principles of Distributed Computing (PODC 1990)*, pages 377–408. ACM, 1990.

[MP92] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer, 1992.

[MP95] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safty*. Springer, 1995.

[MS95] D.E. Muller and P.E. Schupp. Simulating alternating tree automata by nondeterministic automata: New results and new proofs of the theorems of Rabin, McNaughton, and Safra. *Information and Computation*, 141(1-2):69–107, 1995.

[NT00] K.S. Namjoshi and R.J. Trefler. On the completeness of compositional reasoning. In *Proceedings of the 12th International Conference on Computer-Aided Verification (CAV 2000), LNCS 1855*, pages 139–153. Springer, 2000.

[Pit06] N. Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. In *Proceedings of the 21st Annual IEEE Symposium on Logic in Computer Science (LICS 2006)*, pages 255–264. IEEE Computer Society, 2006.

[RF] S. Rodger and T. Finley. JFLAP. http://www.jflap.org/.

[Saf88] 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. IEEE Computer Society, 1988.

[SB00] F. Somenzi and R. Bloem. Efficient Büchi automata from LTL formulae. In *Proceedings of the 12th International Conference on Computer-Aided Verification (CAV 2000), LNCS 1855*, pages 248–263. Springer, 2000.

[Sha00] N. Shankar. Combining model checking and theorem proving through symbolic analysis. In *Proceedings of the 11th International Conference on Concurrency Theory (CONCUR 2000), LNCS 1877*, pages 1–16. Springer, 2000.

[Sis83] A.P. Sistla. *Theoretical Issues in the Design and Verification of Distributed Systems.* PhD thesis, Harvard University, 1983.

[ST03] R. Sebastiani and S. Tonetta. More deterministic vs. smaller Büchi automata for efficient LTL model checking. In *Proceedings of the 12th Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME 2003), LNCS 2860*, pages 126–140. Springer, 2003.

[SVW87] A.P. Sistla, M.Y. Vardi, and P. Wolper. The complementation problem for Büchi automata with applications to temporal logic. *Theoretical Computer Science*, 49:217–237, 1987.

[TCT+07] Y.-K. Tsay, Y.-F. Chen, M.-H. Tsai, K.-N. Wu, and W.-C. Chan. GOAL: A graphical tool for manipulating Büchi automata and temporal formulae. In *Proceedings of the 13th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2007), LNCS 4424*, pages 466–471. Springer, 2007.

[TCT+08] Y.-K. Tsay, Y.-F. Chen, M.-H. Tsai, W.-C. Chan, and C.-J. Luo. GOAL extended: Towards a research tool for omega automata and temporal logic. In *Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2008), LNCS 4963*, pages 346–350. Springer, 2008.

[TCTL07] M.-H. Tsai, W.-C. Chan, Y.-K. Tsay, and C.-J. Luo. Full PTL to Büchi automata translation for on-the-fly model checking. Manuscript, 2007.

[TCW06] 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 (affiliated with FM 2006), 2006.

[Tho99] W. Thomas. Complementation of Büchi automata revisited. In *Jewels are Forever, Contributions on Theoretical Computer Science in Honor of Arto Salomaa*, pages 109–120. Springer, 1999.

[Tsa00] Y.-K. Tsay. Compositional verification in linear-time temporal logic. In *Proceedings of the 3rd International Conference on Foundations of Software Science and Computation Structures (FOSSACS 2000), LNCS 1784*, pages 344–358. Springer, 2000.

[VW86] 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. IEEE Computer Society, 1986.

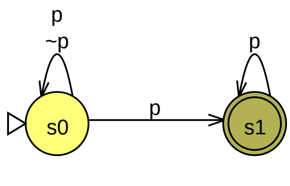
[Wol83] P. Wolper. Temporal logic can be more expressive. *Information and Control*, 56:72–99, 1983.

A. Büchi and Other ω-automata

Büchi automata are the most commonly used type of ω-automata, which extend finite-state automata to infinite words. An ω-automaton accepts an infinite word if and only if there exists a run of the automaton on the word that follows some repetition patterns prescribed by the acceptance condition of the automaton. Formally, an ω-automaton is a quintuple $\langle \Sigma, Q, \delta, q_0, Acc \rangle$:

- Σ is the finite alphabet.
- Q is the finite set of states.
- $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation.
- $q_0 \in Q$ is the initial state.
- Acc is the acceptance condition. Different acceptance conditions give rise to different types of ω-automata.

The automata as defined are *nondeterministic*. An automaton is *deterministic* if, for all $a \in \Sigma$, $q_1, q_2, q_3 \in Q$, $(q_1, a, q_2) \in \delta$ and $(q_1, a, q_3) \in \delta$ imply $q_2 = q_3$. A run of an ω-automaton on an infinite word $a_0 a_1 a_2 \cdots \in \Sigma^\omega$ is an infinite sequence of states $q_0 q_1 q_2 \cdots \in Q^\omega$ such that, for every $i \geq 0$, $(q_i, a_i, q_{i+1}) \in \delta$. Let $\inf(\rho)$ be the set of states that appear infinitely many times (or infinitely often) in the run ρ.



(a) A Büchi automaton.

$$\overbrace{pppp{\sim}ppp{\sim}p\ldots pp{\sim}pp{\sim}p}^{\text{some length } i}\overbrace{pppppppppp\ldots}^{p\text{ always holds}}$$

(b) A word accepted by the automaton.

Fig. 8. A Büchi automaton and a word accepted by the automaton.

A.1. Büchi automata

The acceptance condition of a Büchi automaton is defined by a set of accepting states. A word is accepted by a Büchi automaton if and only if there exists a run of the automaton on the word that passes through at least one accepting state infinitely often. Formally, the acceptance condition of a Büchi automaton is a set $F \subseteq Q$. A run ρ is accepting if $\inf(\rho) \cap F \neq \emptyset$.

In Figure 8(a) is a (nondeterministic) Büchi automaton intended for the property "eventually always p" (which cannot be specified by a deterministic Büchi automaton). Here, the alphabet is {p, ~p}. Figure 8(b) shows an infinite word accepted by the automaton (after the finite prefix of length i, property p always holds). This automaton has a run $\rho = \underbrace{s_0 s_0 s_0 \cdots s_0 s_0}_{i+1} s_1 s_1 s_1 \cdots$ on the word. Since ρ contains infinitely many s_1, ρ is an accepting run and the word is accepted by the automaton. Note that $\rho' = s_0 s_0 s_0 \cdots$ is also a run of the automaton on the same word, but it is not an accepting run because it does not contain even an occurrence of s_1.

Büchi automata recognize ω-regular languages, the infinite-word version of regular languages. We shall introduce some other variants of ω-automata: generalized Büchi automata, Muller automata, Rabin automata, Streett automata, and Parity automata. All these variants of ω-automata, except deterministic Büchi automata, define ω-regular languages and are expressively equivalent.

A.2. Generalized Büchi automata

The acceptance condition of a generalized Büchi automaton is a set of acceptance sets $\{F_1, F_2, \cdots, F_m\}$, where $F_i \subseteq Q$. A word is accepted by a generalized Büchi automaton if and only if there exists a run of the automaton on the word that infinitely often passes through at least one accepting state from each acceptance set. In other words, a run ρ is accepting if, for all i, $\inf(\rho) \cap F_i \neq \emptyset$. Generalized Büchi automata are frequently used as an intermediary in temporal formula to Büchi automaton translation algorithms.

Figure 9 shows an example of generalized Büchi automaton. Here, the alphabet is {pq, p~q, ~pq, ~p~q}. The automaton accepts those words where both p and q hold infinitely often, but not necessarily at the same time. As can be seen from the figure, if a run contains infinitely many s_1, then some transition labeled pq is taken infinitely often, which implies p and q hold infinitely often. Otherwise, an accepting run should contain infinitely many s_2 and s_3 (as dictated by the acceptance condition $\{\{s_1, s_2\}, \{s_1, s_3\}\}$). A transition labeled with q is the only path to visit s_2 and one labeled with p is the only path to visit s_3. Therefore, a run containing infinitely many s_2 and s_3 will enforce that in the input word p holds infinitely often and so does q.

A.3. Other ω-automata

- **Rabin automata**: The acceptance condition of a Rabin automaton is a set of acceptance pairs (pairs of sets of states) $\{(E_1, F_1), (E_2, F_2), \cdots, (E_m, F_m)\}$, where $E_i, F_i \subseteq Q$. A run ρ is accepting if, for some i, $\inf(\rho) \cap E_i = \emptyset$ and $\inf(\rho) \cap F_i \neq \emptyset$.
- **Streett automata**: The acceptance condition of a Streett automaton is also a set of acceptance pairs $\{(E_1, F_1), (E_2, F_2), \cdots, (E_m, F_m)\}$, where $E_i, F_i \subseteq Q$. A run ρ is accepting if, for all i, $\inf(\rho) \cap E_i \neq \emptyset$ or $\inf(\rho) \cap F_i = \emptyset$.

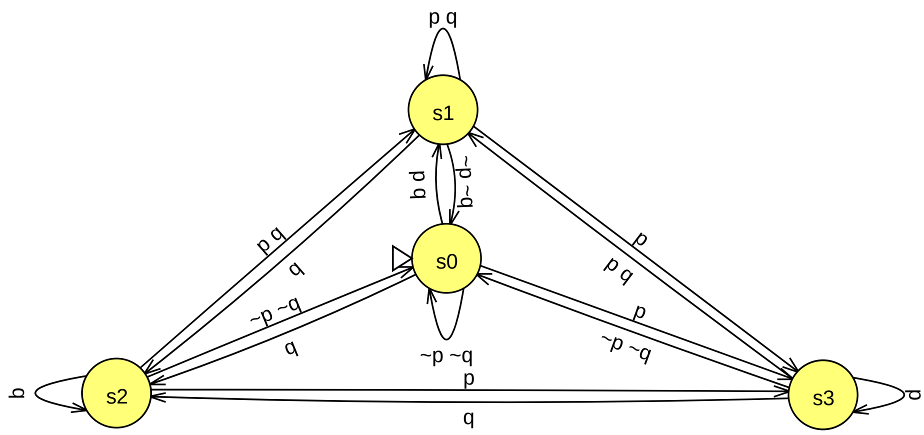


Fig. 9. A generalized Büchi automaton with $\{\{s_1, s_2\}, \{s_1, s_3\}\}$ as the set of acceptance sets.

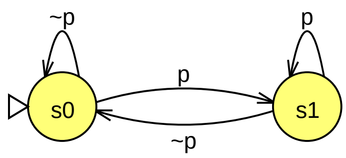


Fig. 10. A Rabin automaton with $\{(\{s_0\}, \{s_1\})\}$ as the set of acceptance pairs.

- **Muller automata**: The acceptance condition of a Muller automaton is a set of acceptance sets $\mathcal{F} = \{F_1, F_2, \cdots, F_m\}$, where $F_i \subseteq Q$. A run ρ is accepting if $\inf(\rho) \in \mathcal{F}$.
- **Parity automata**: The acceptance condition of a Parity automaton is a mapping $c : Q \to \mathbb{N}$. A run ρ is accepting if $\min\{c(q) \mid q \in \inf(\rho)\}$ is even.

Intuitively, given a Rabin acceptance pair (E, F), set E defines the set of states that should be visited only *finitely* many times while set F defines the set of states that should be visited *infinitely* many times. An accepting run satisfies at least one of the Rabin acceptance pairs. The Streett acceptance condition is dual of the Rabin condition. A run is accepting in a Streett automaton if and only if it is not accepting in a Rabin automaton with the same structure and acceptance pairs. A run of a Muller automaton is accepting if and only if the set of states visited infinitely often equals one of the acceptance sets. A parity automaton assumes each state has a parity number. A run of a Parity automaton is accepting if and only if the smallest parity number that is visited infinitely often is even.

In Figure 10 is a deterministic Rabin automaton recognizing "eventually always p". In this automaton, $(\{s_0\}, \{s_1\})$ is the only acceptance pair, which forces every accepting run to end with the infinite sequence $s_1 s_1 s_1 s_1 \cdots$. Given a Rabin automaton, a Streett automaton accepting the complement language of the Rabin automaton can be easily obtained. For example, a Streett automaton recognizing the complement language of the preceding Rabin automaton, namely "not eventually always p", can be obtained by interpreting the acceptance pairs as Streett acceptance condition. Under this interpretation, a run is accepting if and only if it does not end with the infinite sequence $s_1 s_1 s_1 s_1 \cdots$. Because of the convenience in getting a complement automaton (from Rabin to Streett or vice versa), Rabin and Streett automata were used by Safra [Saf88] as the intermediaries for complementing a Büchi automaton.

There are even other variants of ω-automata. For further information, we refer the reader to the book by Grädel *et al.* [GTW02].

B. Linear Temporal Logic

B.1. PTL.

Propositional (Linear) Temporal Logic (PTL) formulae are constructed by applying *boolean* and *temporal* operators to atomic propositions, or boolean variables, drawn from a predefined universe. Temporal operators are classified into future operators and past operators. Future operators include $\bigcirc$ (next), $\diamond$ (eventually), $\square$ (always), $\mathcal{U}$ (until), and $\mathcal{W}$ (wait-for). Past operators include $\circleddash$ (before), $\ominus$ (previous), $\diamondminus$ (once), $\boxminus$ (so-far), $\mathcal{S}$ (since), and $\mathcal{B}$ (back-to).

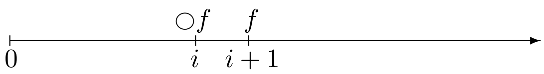
Syntax: Let V be a set of boolean variables. PTL formulae are defined inductively as follows:

- Every variable $p \in V$ is a PTL formula.
- If f and g are PTL formulae, then so are $\neg f$, $f \vee g$, $f \wedge g$, $\bigcirc f$, $\diamond f$, $\square f$, $f\,\mathcal{U}\,g$, $f\,\mathcal{W}\,g$, $\circleddash f$, $\ominus f$, $\diamondminus f$, $\boxminus f$, $f\,\mathcal{S}\,g$, and $f\,\mathcal{B}\,g$. ($\neg f \vee g$ is also written as $f \rightarrow g$ and $(f \rightarrow g) \wedge (g \rightarrow f)$ as $f \leftrightarrow g$.)

Semantics: A PTL formula is interpreted over an infinite sequence of states $\sigma = s_0 s_1 s_2 \cdots$, relative to a position in that sequence. A state is a subset of V, containing exactly those variables that evaluate to true in that state. If each possible subset of V is treated as a symbol, then a sequence of states can also be viewed as an infinite word over 2^V. The semantics of PTL in terms of $(\sigma, i) \models f$ (f holds at the i-th position of σ) is given below. We say that a sequence σ satisfies a PTL formula f or σ is a model of f, denoted $\sigma \models f$, if $(\sigma, 0) \models f$. Two formulae f and g are *equivalent* if all models of f are also models of g and vice versa.

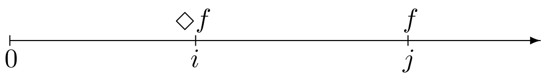
- For a boolean variable p,
  - $(\sigma, i) \models p \iff p \in s_i$
- For boolean operators,
  - $(\sigma, i) \models \neg f \iff (\sigma, i) \models f$ does not hold
  - $(\sigma, i) \models f \vee g \iff (\sigma, i) \models f$ or $(\sigma, i) \models g$
  - $(\sigma, i) \models f \wedge g \iff (\sigma, i) \models f$ and $(\sigma, i) \models g$
- For future temporal operators,
  - $(\sigma, i) \models \bigcirc f \iff (\sigma, i+1) \models f$

    That is, $\bigcirc f$ holds at position i if and only if f holds at position $i+1$, as visualized below.



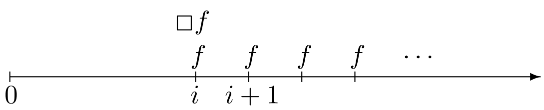
  - $(\sigma, i) \models \diamond f \iff$ for some $j \geq i$, $(\sigma, j) \models f$

    $\diamond f$ holds at position i if and only if f holds at some position $j \geq i$.



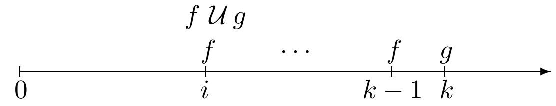
  - $(\sigma, i) \models \square f \iff$ for all $j \geq i$, $(\sigma, j) \models f$

    $\square f$ holds at position i if and only if f holds at every position $j \geq i$; note that f also holds at position i.



– $(\sigma, i) \models f\, \mathcal{U}\, g \iff$ for some $k \geq i$, $(\sigma, k) \models g$ and for all j, $i \leq j < k$, $(\sigma, j) \models f$

$f\, \mathcal{U}\, g$ holds at position i if and only if, for some $k \geq i$, g holds at position k and f holds at every position j, $i \leq j < k$.

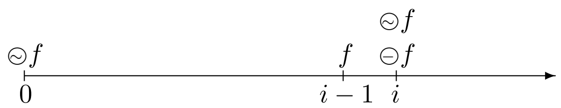


– $(\sigma, i) \models f\, \mathcal{W}\, g \iff$ (for some $k \geq i$, $(\sigma, k) \models g$ and for all j, $i \leq j < k$, $(\sigma, j) \models f$) or (for all $j \geq i$, $(\sigma, j) \models f$)

$f\, \mathcal{W}\, g$ holds at position i if and only if $f\, \mathcal{U}\, g$ or $\Box f$ holds at position i.

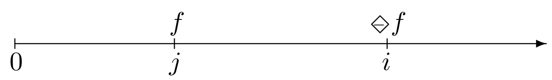
- For past temporal operators,
  – $(\sigma, i) \models \tilde{\ominus} f \iff i = 0$ or $(\sigma, i-1) \models f$
  – $(\sigma, i) \models \ominus f \iff i > 0$ and $(\sigma, i-1) \models f$

  For $i > 0$, $\tilde{\ominus} f$ or $\ominus f$ holds at position i if and only if f holds at position $i-1$. The difference between $\tilde{\ominus} f$ and $\ominus f$ occurs at position 0. $\tilde{\ominus} f$ always holds at position 0, where $\ominus f$ never holds.



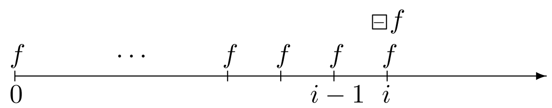
  – $(\sigma, i) \models \diamondminus f \iff$ for some j, $0 \leq j \leq i$, $(\sigma, j) \models f$

  $\diamondminus f$ holds at position i if and only if f holds at some position j, $0 \leq j \leq i$.



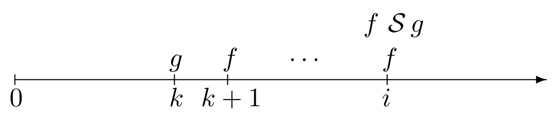
  – $(\sigma, i) \models \boxminus f \iff$ for all j, $0 \leq j \leq i$, $(\sigma, j) \models f$

  $\boxminus f$ holds at position i if and only if f holds at every position j, $0 \leq j \leq i$.



  – $(\sigma, i) \models f\, \mathcal{S}\, g \iff$ for some k, $0 \leq k \leq i$, $(\sigma, k) \models g$ and for all j, $k < j \leq i$, $(\sigma, j) \models f$

  $f\, \mathcal{S}\, g$ holds at position i if and only if for some k, $0 \leq k \leq i$, g holds at position k and f holds at every position j, $k < j \leq i$.



– $(\sigma, i) \models f \, \mathcal{B} \, g \iff$ (for some k, $0 \leq k \leq i$, $(\sigma, k) \models g$ and for all j, $k < j \leq i$, $(\sigma, j) \models f$) or (for all j, $0 \leq j \leq i$, $(\sigma, j) \models f$)

  $f \, \mathcal{B} \, g$ holds at position i if and only if $f \, \mathcal{S} \, g$ or $\boxminus f$ holds at position i.

B.2. QPTL

Quantified Propositional Temporal Logic (QPTL) is PTL extended with quantification over boolean variables (so, every PTL formula is also a QPTL formula):

- If f is a QPTL formula and $x \in V$, then $\forall x \colon f$ and $\exists x \colon f$ are QPTL formulae.

Let $\sigma = s_0 s_1 \cdots$ and $\sigma' = s'_0 s'_1 \cdots$ be two sequences of states. We say that σ' is a x-variant of σ if, for every $i \geq 0$, s'_i differs from s_i at most in the valuation of x, i.e., the symmetric set difference of s'_i and s_i is either $\{x\}$ or empty. The semantics of QPTL is defined by extending that of PTL with additional semantic definitions for the quantifiers:

- For the quantifiers,
  – $(\sigma, i) \models \exists x \colon f \iff (\sigma', i) \models f$ for some x-variant σ' of σ
  – $(\sigma, i) \models \forall x \colon f \iff (\sigma', i) \models f$ for all x-variant σ' of σ

Let us examine the defining QPTL formula for "Even p" (p holds at all even positions): $\exists t \colon (t \wedge \Box(t \leftrightarrow \neg \bigcirc t) \wedge \Box(t \rightarrow p))$. Any sequence σ that satisfies this formula has a t-variant σ' satisfying $t \wedge \Box(t \leftrightarrow \neg \bigcirc t) \wedge \Box(t \rightarrow p)$. From $t \wedge \Box(t \leftrightarrow \neg \bigcirc t)$, we can infer that t holds at all even positions but does not hold at any odd position along σ'. The fact that t holds at all even positions and σ' satisfies $\Box(t \rightarrow p)$ forces p to hold at all even positions in σ'. Since the two sequences σ and σ' are a t-variant of each other, p also holds at all even positions in σ. No restrictions on p are imposed at odd positions, however. The variable t does not hold at any odd position, so p may or may not hold at odd positions in σ' and hence in σ.