Hindawi Publishing Corporation
Journal of Applied Mathematics
Volume 2015, Article ID 516104, 10 pages
http://dx.doi.org/10.1155/2015/516104

Research Article

Degeneralization Algorithm for Generation of Büchi Automata Based on Contented Situation

Laixiang Shan,[1] Jun Qin,[2] Mingshi Chen,[3] and Zheng Qin[1]

[1]*School of Software, Tsinghua University, Beijing 100084, China*
[2]*Credit Reference Center, People's Bank of China, Beijing 100800, China*
[3]*Department of Mathematical Sciences, Tsinghua University, Beijing 100084, China*

Correspondence should be addressed to Zheng Qin; qingzh@mail.tsinghua.edu.cn

Received 7 September 2014; Revised 15 December 2014; Accepted 16 December 2014

Academic Editor: Carlos Conca

We present on-the-fly degeneralization algorithm used to transform generalized Büchi automata (GBA) into Büchi Automata (BA) different from the standard degeneralization algorithm. Contented situation, which is used to record what acceptance conditions are satisfiable during expanding LTL formulae, is attached to the states and transitions in the BA. In order to get the deterministic BA, the Shannon expansion is used recursively when we expand LTL formulae by applying the tableau rules. On-the-fly degeneralization algorithm is carried out in each step of the expansion of LTL formulae. Ordered binary decision diagrams are used to represent the BA and simplify LTL formulae. The temporary automata are stored as syntax directed acyclic graph in order to save storage space. These ideas are implemented in a conversion algorithm used to build a property automaton corresponding to the given LTL formulae. We compare our method to previous work and show that it is more efficient for four sets of random formulae generated by LBTT.

1. Introduction

Model checking [1] is a formal verification technique used to check whether a model of the system verifies some desired properties for software or hardware systems. In order to verify whether the system satisfies some properties, a common method is to use linear temporal logic (LTL) model checking. When the given property is expressed in an LTL formula, the model checker usually transforms the negation of the LTL formula into a Büchi automaton (BA), builds the product of this BA with the system described as an automaton, and checks the emptiness of the product automaton. The size of the product automaton is usually exponential in the size of the system automaton and property automaton, because it is a Cartesian product of the system automaton and property automaton. If there are too many states and transitions, the product automaton will get too big to be verified in the available time. Generating a smaller and more deterministic property automaton in less time contributes to improving the efficiency of model checking.

There are many outstanding conversion tools that implement the translation from an LTL formula to a BA effectively. Babiak et al. [2] proposed a series of improvement measures to improve performance of the conversion algorithm presented by Gastin and Oddoux [3] and implemented a new conversion tool, LTL3BA, which translates an LTL formula into a very weak alternating automaton (VWAA) with a co-Büchi accepting condition. VWAA is then translated into a transition based generalized Büchi automata (TGBA). Finally, TGBA is degeneralized into a BA. The time complexity of the alternation removal is $\mathcal{O}(n2^n)$, which is the same magnitude as alternation removal of tableau-based algorithm [4]. Duret-Lutz [5] introduced many improvements to improve performance of the algorithm proposed by Couvreur [6]. These improvements have been applied in Spot, which is a C++ library for model checking. Spot uses TGBA to express LTL formulae into automata. Gerth et al. [7] proposed a classic algorithm that translates an LTL formula into a generalized Büchi automaton (GBA). This algorithm is a tableau-based translation method in on-the-fly fashion and has been applied in Spin [8].

In order to obtain a BA from the given LTL formula, [2, 3, 5, 7] involve the intermediate automata (GBA or TGBA), perform simplification on the intermediate automata, and transform the intermediate automata into BA finally. Clarke et al. presented a standard degeneralization algorithm used to transform GBA into BA in Section 9.2.2 of [1]. This standard degeneralization algorithm is also adapted to transform a TGBA into a BA [3, 9]. Furthermore, Duret-Lutz proposed a better degeneralization algorithm based on the standard degeneralization algorithm in Section 4.2.2 of [5], which is related to the order in which the corresponding BDD variables were declared. Babiak et al. [10] presented the SCC-based degeneralization including many improvements to the standard degeneralization algorithm used to transform a TGBA into an equivalent BA. Chatterjee et al. [11] proposed the definition of the degeneralization index applied to transform the automaton with generalized Rabin pairs into a Rabin automaton.

However, the standard degeneralization algorithm is used to transform a GBA or a TGBA into a BA, only when the expansion of LTL formulae is finished. We say that the standard degeneralization algorithm is a kind of postdegeneralization algorithm. The intermediate automata are needed to record the expansion of LTL formulae in the use of standard degeneralization algorithm. The standard degeneralization algorithm can transform a TGBA with n states and m acceptance conditions into an equivalent BA with one acceptance condition and at most nm states [12]. We have to search $m!(m+1)$ possible degeneralizations.

In this paper, we present on-the-fly degeneralization algorithm that is used to transform a GBA or a TGBA into an equivalent BA during expanding LTL formulae. We circumvent the intermediate automata and translate an LTL formula to a BA directly. Our method differs from the previous translation algorithms [2, 3, 5, 7] in two ways.

(1) The *contented situation*, which is a set of acceptance conditions, is attached to the states and transitions in the BA. According to the contented situation, we can determine which acceptance conditions are satisfied in the current state or transition.

(2) The process of degeneralization is carried out in each step of the expansion of LTL formulae. LTL formulae can be translated to the BA directly. The intermediate automata are no longer needed.

Our research focuses on an efficient conversion algorithm producing a BA corresponding to an LTL formula directly. The contented situation is attached to the states and transitions in the BA in order to track whether the acceptance condition is satisfiable. The BA is described by ordered binary decision diagrams (OBDDs) and stored as syntax directed acyclic diagram (DAG). On-the-fly degeneralization algorithm is used in order to degeneralize GBA into BA during the expansion. The BA simplification is adopted in the algorithm in order to gain reduction on the size of the result automaton. In order to get the deterministic BA, the Shannon expansion is used recursively when we expand LTL formulae by applying the tableau rules. These measures cause a lot of improvement on the efficiency of the algorithm, especially when expanding the formulae containing a large amount of $\{\mathbf{U}, \mathbf{R}, \mathbf{F}, \mathbf{G}\}$-subformulae.

The rest of this paper is organized as follows. In Section 2, we provide preliminary notions used in this paper. Then, we describe the main ideas of our approach in Section 3. Overview of the algorithm is introduced in Section 4. In Section 5, a comparison between our method and previous works is presented. Finally, Section 6 closes the paper with conclusions.

2. Preliminaries

LTL is usually used to describe system constraints in formal method, which is a modal temporal logic with modalities referring to time. Let AP represent a finite set of atomic propositions. Let 2^{AP} represent the set of subsets of AP. Let $2^{2^{\mathrm{AP}}}$ represent the set of propositional formulae induced by AP.

Definition 1 (syntax of LTL formulae). An LTL formula is usually composed of atomic propositions ($p \in \mathrm{AP}$), proposition constants ($\top$ (True) and $\bot$ (False)), the logical operators ($\neg$ (not), $\wedge$ (and), and $\vee$ (or)), and the temporal modal operators ($\mathbf{X}$ (Next), $\mathbf{U}$ (Until), $\mathbf{R}$ (Release), $\mathbf{G}$ (Always), and $\mathbf{F}$ (Eventually)). Formally, the syntax of LTL formulae is defined inductively as follows:

(i) p, $\top$, and $\bot$ are LTL formulae, respectively;

(ii) $\neg\varphi$, $\varphi \wedge \psi$, $\varphi \vee \psi$, $\mathbf{X}\varphi$, $\mathbf{F}\varphi$, $\mathbf{G}\varphi$, $\varphi\mathbf{U}\psi$, and $\varphi\mathbf{R}\psi$ are LTL formulae, if φ and ψ are LTL formulae, respectively.

$\mathbf{R}$-formula is the dual of $\mathbf{U}$-formula. $\{\mathbf{R}, \mathbf{F}, \mathbf{G}\}$-formula can be translated to $\mathbf{U}$-formula by the following identities:

(i) $\varphi\mathbf{R}\psi \equiv \neg(\neg\varphi\mathbf{U}\neg\psi)$;

(ii) $\mathbf{F}\varphi \equiv \top\mathbf{U}\varphi$;

(iii) $\mathbf{G}\varphi \equiv \bot\ \mathbf{R}\varphi \equiv \neg\mathbf{F}(\neg\varphi) \equiv \neg(\top\mathbf{U}\neg\varphi)$.

Definition 2 (Kripke structure). A Kripke structure is a tuple $M = (Q, \delta, L)$ where Q is a finite set of states. δ is $\delta \subseteq Q \times Q$, a transition relation between states. L is $Q \to \mathrm{AP}$, labeling of the states.

Definition 3 (semantics of LTL formulae). Let M be a Kripke structure, and let $\xi = \xi[0]\xi[1]\xi[2]\cdots \in (2^{\mathrm{AP}})^{\omega}$ be an infinite word in M. $\xi_i = \xi[i]\xi[i+1]\xi[i+2]\cdots$ denotes the suffix starting at letter $\xi[i]$. The semantics of LTL formulae is defined inductively as follows:

(i) $M, \xi \vDash \top$;

(ii) $M, \xi \vDash p$ iff $p \in \xi[0]$, for $p \in \mathrm{AP}$;

(iii) $M, \xi \vDash \neg\varphi$ iff $\neg(\xi \vDash \varphi)$;

(iv) $M, \xi \vDash \varphi \wedge \psi$ iff $\xi \vDash \varphi$ and $\xi \vDash \psi$;

(v) $M, \xi \vDash \varphi \vee \psi$ iff $\xi \vDash \varphi$ or $\xi \vDash \psi$;

(vi) $M, \xi \vDash \mathbf{X}\varphi$ iff $\xi_1 \vDash \varphi$;

(vii) $M, \xi \vDash \mathbf{F}\varphi$ iff $\exists i \geq 0, \xi_i \vDash \varphi$;

(viii) $M, \xi \vDash \mathbf{G}\varphi$ iff $\forall i \geq 0, \xi_i \vDash \varphi$;

(ix) $M, \xi \vDash \varphi \mathbf{U} \psi$ iff $\exists j \geq 0, \xi_j \vDash \psi$ and $\forall 0 \leq i < j, \xi_i \vDash \varphi$;

(x) $M, \xi \vDash \varphi \mathbf{R} \psi$ iff $\exists j \geq 0, \xi_j \vDash \varphi$ and $\forall 0 \leq i < j, \xi_i \vDash \varphi$ and $\xi_i \vDash \psi$.

Remark 4. Every LTL formula can be rewritten as an equivalent LTL formula in negation normal form (NNF), where operator $\neg$ occurs only immediately in front of atomic propositions and $\neg$, $\wedge$, and $\vee$ are the only allowed Boolean connectives. In this paper, we consider only such formulae. The NNF formula can be translated to an equivalent LTL formula by the following identities:

$$\begin{aligned} \neg(\varphi \wedge \psi) &\equiv \neg\varphi \vee \neg\psi, \\ \neg(\varphi \vee \psi) &\equiv \neg\varphi \wedge \neg\psi, \\ \neg\neg\varphi &\equiv \varphi, \\ \neg\mathbf{X}\varphi &\equiv \mathbf{X}\neg\varphi, \\ \neg\mathbf{F}\varphi &\equiv \mathbf{G}\neg\varphi, \\ \neg(\varphi \mathbf{U} \psi) &\equiv \neg\varphi \mathbf{R} \neg\psi, \\ \neg(\varphi \mathbf{R} \psi) &\equiv \neg\varphi \mathbf{U} \neg\psi, \\ \neg\mathbf{G}\varphi &\equiv \mathbf{F}\neg\varphi. \end{aligned} \tag{1}$$

Definition 5 (Büchi automata). Büchi automaton is a kind of ω-automata in which acceptance conditions are carried by the states. It is also called state-based Büchi automata that is a tuple $\mathscr{A} = \langle Q, \Sigma, \delta, I, F \rangle$, where Q is a finite set of states. Σ is a finite input alphabet, $\Sigma = 2^{\mathrm{AP}}$. $\delta : Q \times \Sigma \to 2^{Q}$ is a transition function. $I \subseteq Q$ is a set of initial states. $F \subseteq Q$ is a set of acceptance states. Let $\xi = \xi[0]\xi[1]\xi[2] \cdots \in \Sigma^{\omega}$ be an infinite word of $\mathscr{A}$. An infinite sequence $\pi = (q_0, \xi[0], q_1)(q_1, \xi[1], q_2) \cdots \in \delta^{\omega}$ is a run of $\mathscr{A}$, where $q_0 \in I$ is an initial state and $q_{i+1} \in \delta(q_i, \xi[i])$ for all $i \geq 0$. $\mathrm{Run}(\mathscr{A})$ denotes the set of all runs of $\mathscr{A}$. Let $\mathrm{Inf}_Q(\pi)$ denote the set of states that appear infinitely often in π. The run π is accepted by $\mathscr{A}$, if and only if $\exists \pi \in \mathrm{Run}(\mathscr{A})$ with $\mathrm{Inf}_Q(\pi) \cap F \neq \emptyset$. An infinite word ξ is accepted by $\mathscr{A}$, if some run of $\mathscr{A}$ over ξ is accepted.

Definition 6 (transition based generalized Büchi automata). Transition based generalized Büchi automaton (TGBA) is a Büchi automaton in which the set of subsets of acceptance conditions are carried by the transitions. It can be defined as a tuple $\mathscr{T} = \langle Q, \mathrm{AP}, \delta, I, F \rangle$, where Q is a finite set of states. AP is a finite set of atomic propositions. $\delta \subseteq Q \times 2^{2^{\mathrm{AP}}} \times 2^{F} \times Q$ is a transition relation, where each transition is labeled by a Boolean formula and a set of acceptance conditions. $I \subseteq Q$ is a set of initial states. $F = \{f_1, f_2, \ldots, f_m\}$ is a finite set of acceptance conditions, m is the number of acceptance conditions, and 2^{F} is the set of the subsets of acceptance conditions. An infinite sequence $\pi = (q_0, l_0, F_0, q_1) \cdots (q_i, l_i, F_i, q_{i+1}) \cdots \in \delta^{\omega}$ is a run of $\mathscr{T}$, where $q_0 \in I$, $l_i \in 2^{2^{\mathrm{AP}}}$, $F_i \in 2^{F}$, and $q_{i+1} \in \delta(q_i, l_i, F_i)$ for all $i \geq 0$. $\mathrm{Run}(\mathscr{T})$ denotes the set of all runs of $\mathscr{T}$. Let $\mathrm{Inf}_\delta(\pi)$ denote the set of transitions that appear infinitely often in π. The run π is accepted by $\mathscr{T}$, if and only if $\forall Z \in F, \exists \pi \in \mathrm{Run}(\mathscr{T})$ with $\mathrm{Inf}_\delta(\pi) \cap Z \neq \emptyset$. The accepting runs of $\mathscr{T}$ visit each acceptance set infinitely often.

Table 1: Tableau rules for LTL formulae.

Formula	1st subformula	2nd subformula
$\varphi \wedge \psi$	$\{\varphi, \psi\}$	
$\varphi \vee \psi$	$\{\varphi\}$	$\{\psi\}$
$\mathbf{X}\varphi$	$\{\mathbf{X}\varphi\}$	
$\varphi \mathbf{U} \psi$	$\{\psi\}$	$\{\varphi, \mathbf{X}(\varphi \mathbf{U} \psi)\}$
$\varphi \mathbf{R} \psi$	$\{\varphi, \psi\}$	$\{\psi, \mathbf{X}(\varphi \mathbf{R} \psi)\}$
$\mathbf{F}\varphi$	$\{\varphi\}$	$\{\mathbf{X}\mathbf{F}\varphi\}$
$\mathbf{G}\varphi$	$\{\varphi, \mathbf{X}\mathbf{G}\varphi\}$	

3. Details of Our Approach

In this section, we introduce our ideas and the details of our approach. The next section gives an overview of the algorithm implementation.

3.1. Tableau Rules. Tableau rules [7] are often used to translate an LTL formula into a BA. The process of translation is as follows. First, an LTL formula φ, which is rewritten as NNF, is defined as the labeling of the initial state. Then, φ is expanded by applying the tableau rules recursively until no $\{\mathbf{U}, \mathbf{R}, \mathbf{F}, \mathbf{G}\}$-subformulae occur at the top level. The expansion of the formula is rewritten as a *cover* by computing its disjunctive normal form (DNF). Each disjunct of the *cover* represents a state of the automaton. All propositional literals represent the label of the states, which are the acceptance conditions satisfied in this state. The $\mathbf{X}$-formulae represent the label of the next state and determine the transitions outcoming from the current state. The transition is given by connecting each state to its successors. The tableau rules for LTL formulae are listed in Table 1. The automata constructed by the tableau rules are state-based Büchi automata.

For instance, the expansions of $\alpha \mathbf{U} \beta$ and $\alpha \mathbf{R} \beta$ are showed as (2). In the *cover* of $\alpha \mathbf{U} \beta$, there are two propositional literals and one $\mathbf{X}$-formula. Consider

$$\begin{aligned} \alpha \mathbf{U} \beta &\Longrightarrow \beta \vee (\alpha \wedge \mathbf{X}(\alpha \mathbf{U} \beta)), \\ \alpha \mathbf{R} \beta &\Longrightarrow (\alpha \wedge \beta) \vee (\beta \wedge \mathbf{X}(\alpha \mathbf{R} \beta)). \end{aligned} \tag{2}$$

By observing the expansion of $\alpha \mathbf{U} \beta$, we find that either (i) β holds and $\alpha \mathbf{U} \beta$ is satisfied or (ii) α holds and $\alpha \mathbf{U} \beta$ should be verified next, until $\alpha \mathbf{U} \beta$ has to be satisfied. Obviously, (ii) is infinite path probably. The expansion of $\alpha \mathbf{R} \beta$ is similar to $\alpha \mathbf{U} \beta$. Intuitively, $\mathbf{F}\alpha \Rightarrow \alpha \vee \mathbf{X}\mathbf{F}\alpha$ and $\mathbf{G}\alpha \Rightarrow \alpha \wedge \mathbf{X}\mathbf{G}\alpha$ product infinite path also.

Although the method based on tableau rules can work well, the only challenging problem is how to avoid the unwanted infinite path. Because tableau rules cannot fully characterize $\{\mathbf{U}, \mathbf{R}, \mathbf{F}, \mathbf{G}\}$-formulae, each infinite path cannot identify models of the formula according to a graph which is constructed by the sole expansion law. In order to cope with

(1) $\mathcal{S}(q_0) = \emptyset$; $(*)$
(2) **if** $\exists\varphi \in F$ *with* $\varphi \in E(q_i)$ *and* $l(t_i) \models^* \varphi$ **then**
$\mathcal{S}(t_i) = \mathcal{S}(q_i) \cup \varphi$
else
$\mathcal{S}(t_i) = \mathcal{S}(q_i)$;
endif $(**)$
(3) **if** $\exists\varphi \in F$ *with* $\varphi \in E(q_{i+1})$ *and* $\varphi \notin \mathcal{S}(t_i)$ **then**
$\mathcal{S}(q_{i+1}) = \mathcal{S}(t_i)$
else
$\mathcal{S}(q_{i+1}) = \emptyset$;
endif; $(***)$

Algorithm 1

the infinite path, the *promise* has been presented in [13]. In this paper, a new approach, *contented situation*, is presented. We attach contented situation to the states and transitions in the automata during expanding LTL formulae. Although a sequence of BA is infinite probably, it is accepted as long as the formula is satisfied infinite often along the sequence. We record the acceptance conditions that are satisfied along the sequence. If all the acceptance conditions are satisfied, then we arrive at the final state. It guarantees the sequence of the $\{\mathbf{U}, \mathbf{R}, \mathbf{F}, \mathbf{G}\}$-formulae cannot be postponed infinitely.

3.2. Contented Situation. The *contented situation* is a set of acceptance conditions that are satisfiable along the path of a BA. The *contented situation* differs from the *promise* in two ways: the definition and the purpose. The promise is the opposite of the acceptance conditions [13], while the contented situation is the set of the satisfying acceptance conditions. The promise is attached to the transitions only, while the contented situation is attached to the states or transitions. According to the promise, we can determine which acceptance conditions are not satisfied in the current transition. According to the contented situation, we can determine which acceptance conditions are satisfied in the current state or transition.

Definition 7 (contented situation). A function $\mathcal{S}$ is defined to express the contented situation that is a mapping relationship from the states and transitions to the set of acceptance conditions. The formalized definition of $\mathcal{S}$ is as follows:

$$\mathcal{S} : Q \cup T \longrightarrow 2^F, \tag{3}$$

where Q is a finite set of states, F represents a set of the acceptance conditions, and T is a finite set of transitions.

$\mathcal{S}$ indicates which acceptance conditions are satisfiable along a run of the automata. $\mathcal{S}$ of the initial state q_0 is defined as $\emptyset$ ($(*)$ of Algorithm 1). q_i is the ith state and t_i is the ith transition in the automata. The recursive calculation of $\mathcal{S}(t_i)$ and $\mathcal{S}(q_i)$ is shown as $(**)$-$(***)$ in Algorithm 1, where $E(q_i)$ represents the eventualities that should be satisfied along the sequence starting from q_i. Let $\pi = (q_1, l(t_1), q_2)(q_2, l(t_2), q_3)\cdots$ be a run in the automata. If $\pi \models \varphi$ and $l(t_i) \in \pi[i]$, then $l(t_i) \models^* \varphi$. $l(t_i)$ represents the label of the transition t_i.

For example, the expansion of $(\alpha\mathbf{U}\beta)\mathbf{U}\gamma$ is showed in Figure 1. The contented situations (in the curly braces) are attached to the states and transitions. $F = \{\alpha\mathbf{U}\beta, (\alpha\mathbf{U}\beta)\mathbf{U}\gamma\}$ and $\mathcal{S}(q_0) = \emptyset$. $\mathcal{S}$ of the transition $t_0 = (q_0, \alpha, q_1)$ is $\emptyset$, because α cannot satisfy $(\alpha\mathbf{U}\beta)\mathbf{U}\gamma$. $\mathcal{S}(q_1) = \emptyset$, because $\nexists\varphi \in F \cap E(q_1)$ s.t. $\varphi \in \mathcal{S}(t_0)$. The formula is expanded recursively until $\mathcal{S}(q_3) = \emptyset$; that is, the final state is reached.

3.3. Computing Deterministic Covers. As said in Section 3.1, the covers of LTL formulae are computed based on the recursive application of the tableau rules and on the subsequent computation of the DNF of the resulting formula. The determinism of the BA can be improved using a trick on the Shannon expansion during the expansion.

For example, the cover of $\mathbf{GF}\varphi \wedge \mathbf{GF}\psi$ is showed in (4). The BA corresponding to this cover has four successor states q_0, q_1, q_2, and q_3 with labels $\{\varphi, \psi\}$, $\{\varphi\}$, $\{\psi\}$, and $\{\top\}$, respectively. The states, q_1, q_2, and q_3, are nondeterministic decision states. Consider

$$\begin{aligned}
&\operatorname{cover}(\mathbf{GF}\varphi \wedge \mathbf{GF}\psi)\\
&\quad = (\varphi \wedge \psi \wedge \mathbf{X}(\mathbf{GF}\varphi \wedge \mathbf{GF}\psi))\\
&\qquad \vee (\varphi \wedge \mathbf{X}(\mathbf{F}\psi \wedge \mathbf{GF}\varphi \wedge \mathbf{GF}\psi))\\
&\qquad \vee (\psi \wedge \mathbf{X}(\mathbf{F}\varphi \wedge \mathbf{GF}\varphi \wedge \mathbf{GF}\psi))\\
&\qquad \vee (\top \wedge \mathbf{X}(\mathbf{F}\varphi \wedge \mathbf{F}\psi \wedge \mathbf{GF}\varphi \wedge \mathbf{GF}\psi)).
\end{aligned} \tag{4}$$

The states, q_1, q_2, and q_3, can be expanded by applying the Shannon expansion. The expansions of q_1, q_2, and q_3 are showed in (5). According to the identical equation $\mathbf{GF}\varphi \equiv (\mathbf{F}\varphi \wedge \mathbf{XGF}\varphi)$, we know $(\mathbf{F}\psi \wedge \mathbf{GF}\varphi \wedge \mathbf{GF}\psi) \equiv (\mathbf{GF}\varphi \wedge \mathbf{GF}\psi)$, $(\mathbf{F}\varphi \wedge \mathbf{GF}\varphi \wedge \mathbf{GF}\psi) \equiv (\mathbf{GF}\varphi \wedge \mathbf{GF}\psi)$, and $(\mathbf{F}\varphi \wedge \mathbf{F}\psi \wedge \mathbf{GF}\varphi \wedge \mathbf{GF}\psi) \equiv (\mathbf{GF}\varphi \wedge \mathbf{GF}\psi)$:

$$\begin{aligned}
&(\varphi \wedge \mathbf{X}(\mathbf{F}\psi \wedge \mathbf{GF}\varphi \wedge \mathbf{GF}\psi))\\
&\quad = (\varphi \wedge \psi \wedge \mathbf{X}(\mathbf{GF}\varphi \wedge \mathbf{GF}\psi))\\
&\qquad \vee (\varphi \wedge \neg\psi \wedge \mathbf{X}(\mathbf{GF}\varphi \wedge \mathbf{GF}\psi)),\\
&(\psi \wedge \mathbf{X}(\mathbf{F}\varphi \wedge \mathbf{GF}\varphi \wedge \mathbf{GF}\psi))\\
&\quad = (\psi \wedge \varphi \wedge \mathbf{X}(\mathbf{GF}\varphi \wedge \mathbf{GF}\psi))
\end{aligned}$$

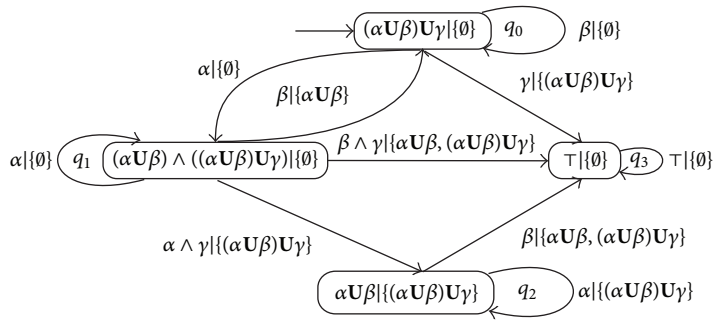


Figure 1: The expansion of $(\alpha \mathbf{U} \beta)\mathbf{U}\gamma$.

$$\begin{aligned}
&\vee (\psi \wedge \neg\varphi \wedge \mathbf{X}(\mathbf{GF}\varphi \wedge \mathbf{GF}\psi)),\\
(\top \wedge \mathbf{X}(\mathbf{F}\varphi \wedge \mathbf{F}\psi \wedge \mathbf{GF}\varphi \wedge \mathbf{GF}\psi)) &= (\varphi \wedge \psi \wedge \mathbf{X}(\mathbf{GF}\varphi \wedge \mathbf{GF}\psi))\\
&\vee (\varphi \wedge \neg\psi \wedge \mathbf{X}(\mathbf{GF}\varphi \wedge \mathbf{GF}\psi))\\
&\vee (\neg\varphi \wedge \psi \wedge \mathbf{X}(\mathbf{GF}\varphi \wedge \mathbf{GF}\psi))\\
&\vee (\neg\varphi \wedge \neg\psi \wedge \mathbf{X}(\mathbf{GF}\varphi \wedge \mathbf{GF}\psi)).
\end{aligned} \tag{5}$$

By combining the same items according to (5), (4) can be rewritten as (6). The BA corresponding to the cover in (6) has four successor states q_0', q_1', q_2', and q_3' with labels $\{\varphi, \psi\}$, $\{\varphi, \neg\psi\}$, $\{\neg\varphi, \psi\}$, and $\{\neg\varphi, \neg\psi\}$, respectively. The states, q_0', q_1', q_2', and q_3', are deterministic decision states. Consequently, deterministic covers give rise to deterministic automata. The TGBA corresponding to $\mathbf{GF}\varphi \wedge \mathbf{GF}\psi$ is showed in Figure 2(a):

$$\begin{aligned}
\operatorname{cover}(\mathbf{GF}\varphi \wedge \mathbf{GF}\psi) &= (\varphi \wedge \psi \wedge \mathbf{X}(\mathbf{GF}\varphi \wedge \mathbf{GF}\psi))\\
&\vee (\varphi \wedge \neg\psi \wedge \mathbf{X}(\mathbf{GF}\varphi \wedge \mathbf{GF}\psi))\\
&\vee (\neg\varphi \wedge \psi \wedge \mathbf{X}(\mathbf{GF}\varphi \wedge \mathbf{GF}\psi))\\
&\vee (\neg\varphi \wedge \neg\psi \wedge \mathbf{X}(\mathbf{GF}\varphi \wedge \mathbf{GF}\psi)).
\end{aligned} \tag{6}$$

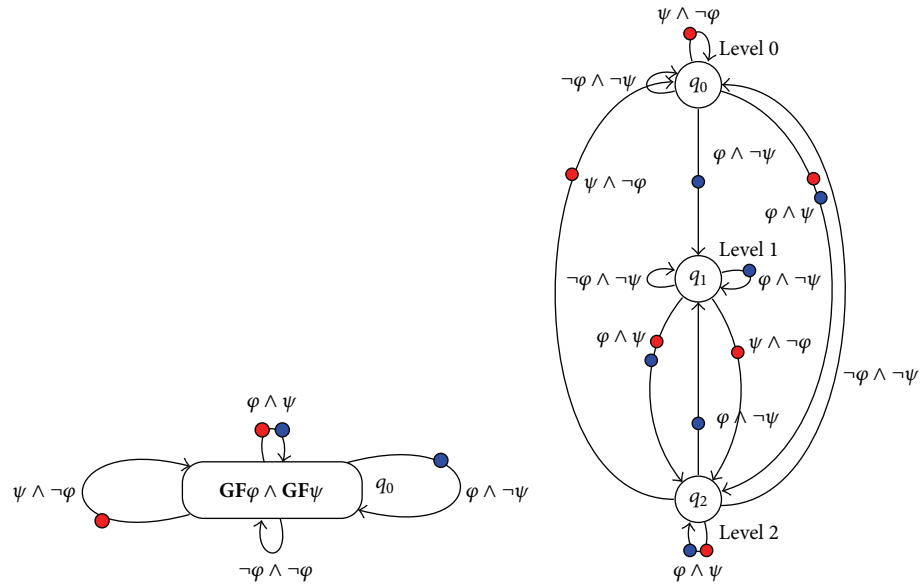
There are 2^n labels to be considered during the expansion by applying the Shannon expansion over n atomic propositions in a BA. However, we remove the invalid nodes and combine equivalent states during the expansion of LTL formulae. It helps to ignore the useless labels. In the practical application, n is usually small enough so that the slowdown incurred by this method has little effect.

3.4. On-the-Fly Degeneralization. The degeneralization is a simple translation from a GBA to a BA. The standard degeneralization algorithm is presented by Clarke et al. [1]. This standard degeneralization algorithm can be used to translate a TGBA into a BA [2, 3, 5]. However, the standard algorithm is a kind of postdegeneralization algorithm; that is to say, it is used to translate a GBA or a TGBA to a BA, only when the expansion of LTL formulae is finished. The intermediate automata (GBA or TGBA) are needed to record the expansion of LTL formulae. For a TGBA, there are $m!(m + 1)$ possible degeneralizations by applying this algorithm. It takes a lot of time and storage space to complete the degeneralization.

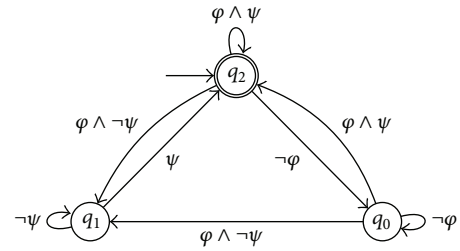
The degeneralization of $\mathbf{GF}\varphi \wedge \mathbf{GF}\psi$ by applying the standard degeneralization algorithm is showed in Figure 2. The details are introduced in Section 4.2.2 of [5]. The TGBA corresponding to $\mathbf{GF}\varphi \wedge \mathbf{GF}\psi$, which is showed in Figure 2(a), has two acceptance conditions $\{\mathrm{Acc}[\mathbf{F}\varphi], \mathrm{Acc}[\mathbf{F}\psi]\}$ that are indicated using colored marker (the blue dot and the red dot) per set. In Figure 2(b), in addition to the first level, the states in each level satisfy at least one acceptance condition. The states in the last level satisfy all the acceptance conditions. In Figure 2(b), all states in level 1 satisfy $\{\mathrm{Acc}[\mathbf{F}\varphi]\}$; all states in level 2 satisfy $\{\mathrm{Acc}[\mathbf{F}\varphi], \mathrm{Acc}[\mathbf{F}\psi]\}$. For example, q_1 in level 1 satisfies $\{\mathrm{Acc}[\mathbf{F}\varphi]\}$, and q_2 in level 2 satisfies $\{\mathrm{Acc}[\mathbf{F}\varphi], \mathrm{Acc}[\mathbf{F}\psi]\}$ in Figure 2(b).

All transitions are added to the degeneralized automata according to which acceptance conditions are satisfied in the current state. In order not to lose the transition, the TGBA is cloned in $m + 1$ levels. The number of transitions in the degeneralized automata is three times the ones of TGBA. This setup guarantees that any accepting run in the degeneralized automata will correspond to an infinite run that visits all acceptance conditions infinitely often in the TGBA. The BA corresponding to $\mathbf{GF}\varphi \wedge \mathbf{GF}\psi$ is generated by removing the invalid states and transitions from degeneralized automata in Figure 2(c). The standard degeneralization algorithm generates many invalid states and transitions (see Figure 2(b)). It takes a lot of time to generate the degeneralized automata and remove the invalid states and transitions. A lot of storage space has to be used to save the temporal data.

In this paper, we present on-the-fly degeneralization algorithm used to translate a GBA or a TGBA to a BA during the expansion of LTL formulae. The degeneralized automata are no longer needed in on-the-fly degeneralization algorithm. Our idea is that the degeneralization is carried out in each step of the expansion. As described in Section 3.2, the contented situation is attached to all states in the BA. According to the contented situation, we know which acceptance conditions are satisfiable in each state. We can add the transitions to the BA according to which acceptance conditions are satisfied in the current state; that is to say,



(a) TGBA corresponding to $\mathbf{GF}\varphi \wedge \mathbf{GF}\psi$ (b) Degeneralization of the TGBA



(c) BA corresponding to $\mathbf{GF}\varphi \wedge \mathbf{GF}\psi$

Figure 2: Degeneralization of a TGBA corresponding to $\mathbf{GF}\varphi \wedge \mathbf{GF}\psi$ using the standard degeneralization algorithm.

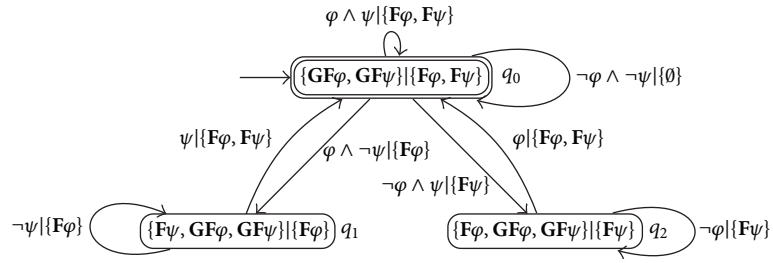
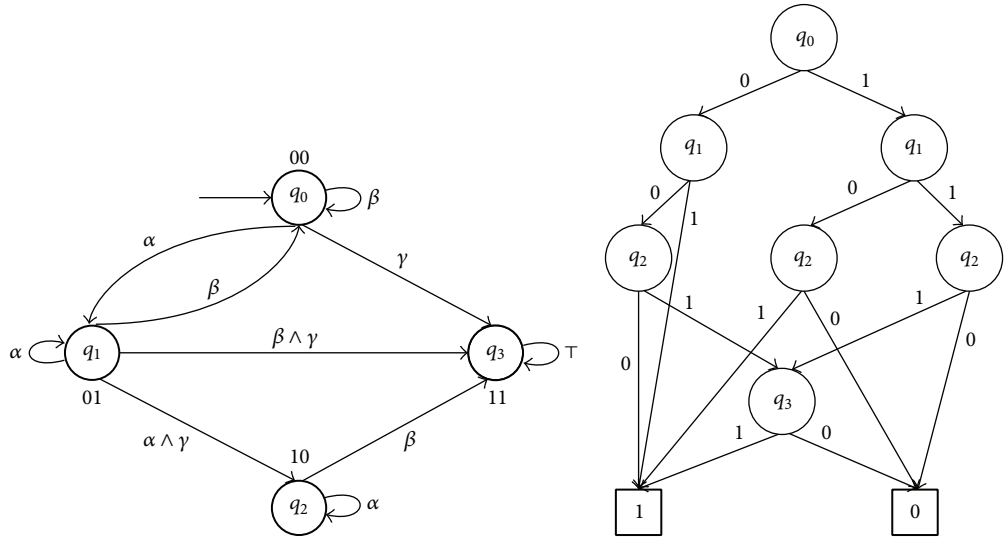


Figure 3: Building a BA corresponding to $\mathbf{GF}\varphi \wedge \mathbf{GF}\psi$ using on-the-fly degeneralization.

the outgoing transitions that carry the acceptance condition f_j are redirected to the next state in which f_j is satisfiable. The outgoing transitions that carry $\emptyset$ are redirected to the initial state. This setup guarantees that any accepting run can see all acceptance conditions infinitely often in the automata.

On-the-fly degeneralization is a process in which the degeneralization process is carried out during the expansion of LTL formulae. Therefore, we can translate an LTL formula to a BA directly. The time used to generate the degeneralized automata and remove redundant states and transitions is saved. In Figure 3, we translate $\mathbf{GF}\varphi \wedge \mathbf{GF}\psi$ to a BA directly by applying on-the-fly degeneralization. The contented situation is attached to the states and transitions. The contented situation of the transition t_i is equal to the contented situation of the destination state of this transition. For example, the outgoing transitions that carry the acceptance condition $\{\mathrm{Acc}[\mathbf{F}\varphi]\}$ are redirected to the next state in which $\{\mathrm{Acc}[\mathbf{F}\varphi]\}$ is satisfiable. The outgoing transitions that carry the acceptance condition $\{\mathrm{Acc}[\mathbf{F}\psi]\}$ are redirected to the next state in which $\{\mathrm{Acc}[\mathbf{F}\psi]\}$ is satisfiable. The outgoing transitions that carry the acceptance condition $\{\mathrm{Acc}[\mathbf{F}\varphi], \mathrm{Acc}[\mathbf{F}\psi]\}$ are redirected to the next state in which $\{\mathrm{Acc}[\mathbf{F}\varphi], \mathrm{Acc}[\mathbf{F}\psi]\}$ is satisfiable. On-the-fly degeneralization is a process in which the standard degeneralization algorithm is carried out in each



(a) The BA corresponding to $(\alpha \mathbf{U} \beta)\mathbf{U}\gamma$ (b) Representing the BA by applying OBDD

Figure 4: Rewriting Büchi automata into an OBDD.

step of expansion. The BA in Figure 3 is equivalent to the BA in Figure 2(c).

Although the BA in Figure 3 is different from the BA in Figure 2(c), they both guarantee that any accepting run visits all acceptance conditions infinitely often in the automata with the same number of states and transitions.

3.5. Representing a BA by OBDD. The BA can be represented by applying ordered binary decision diagrams (OBDDs) [14]. OBDD is a binary decision diagram that has an ordering for some list of variables. For example, we represent the BA corresponding to $(\alpha \mathbf{U} \beta)\mathbf{U}\gamma$ in Figure 1 by applying OBDDs. OBDDs representation is showed in Figure 4.

OBDDs can be regarded as a compressed representation of sets or relations. The checking equivalence is reduced to checking isomorphism between BDDs, when OBDDs are used to represent the BA. If the two Boolean functions have isomorphic representations in OBDDs, they are logically equivalent and can be merged. OBDD is a more concise representation for the cover by merging isomorphic subtrees.

OBDD, which is a rooted, directed acyclic graph, can be stored as a syntax directed acyclic diagram (DAG). The use of a DAG has two advantages. First, less space is needed to store the automata. Second, if the result of computing each subtree is cached, there will be concomitant time savings. Sharing of subformulae also can work across multiple formulae. Because each subformula is unique, a cache is used to record the formula when we are travelling multiple formulae in order to speed up the algorithm. The running time is reduced by applying this kind of data structure.

4. Overview of Algorithm

In the current, many kinds of state-of-the-art algorithm used to translate an LTL formula to a BA can be divided into four main phases: simplifying the formulae, translating an LTL formula to a BA, the BA simplification, and degeneralization of the BA.

In this paper, we focus on phase 2 and phase 3. According to the new approach proposed in the previous section, we have conceived and implemented a new translation tool, called *ltltoba*, which builds a BA from an LTL formula. The contented situation and on-the-fly degeneralization algorithm are used in these tools. Our method differs from the previous translation algorithms [2, 3, 5, 7] in two steps.

(1) Our algorithm circumvents the intermediate automata and translates an LTL formula to a BA directly.

(2) On-the-fly degeneralization is used in our algorithm. Phase 4 is not needed.

4.1. The Data Structure. The basic data structure that the automata graph construction algorithm manipulates is the *Node*, which contains the following fields: **NodeSet**, the set of the unprocessed states; **DisjunctSet**, the set of the disjuncts of the cover; **f**, the input LTL formulae; $\mathbf{q}, \mathbf{q}'$, the state of the automata; **t**, the transition of the automata; **src**, source node of a transition; **dst**, destination node of a transition; **l**, the label of the state or transition; **F**, the set of the acceptance conditions.

In order to describe the BA, the class is defined as shown in Algorithm 2.

4.2. The Main Algorithm. The main algorithm of ltltoba, which is described in Algorithm 3, is used to transform a given LTL formula f to a BA. The idea is very clear; we expand an LTL formula recursively by applying the tableau rules. A simple depth-first-search strategy is used in the expansion operation. The resulting formula is rewritten into a cover by computing its DNF. For each disjunct of the cover, the contented situation of state or transition is

```
class T_BA
{
public:
  void initialize_state(f∅);          // the initial node is defined as f, the contented situation is ∅.
  void add_state(q𝒮(q));     // adding a state to the BA. If a BA has not the state q, q is added to
  the BA directly. If a BA has a state q' with l(q) = l(q'), then 𝒮(q') = 𝒮(q) ∪ 𝒮(q') and q is discarded.
  void add_trans(t𝒮(t));    // adding a transition to the BA. If a BA has not the
  transition t, t is added to the BA directly. If a BA has a transition t' with src(t) = src(t'),
  dst(t) = dst(t') and 𝒮(t) = 𝒮(t'), then label(t') = label(t) ∪ label(t') and t is discarded.
  string search_state(q)          // Searching the state q in the BA.
  If ∃q' ∈ BA s.t. l(q) = l(q'), then q already exists in the BA and return q'.
  Otherwise, return ∅.
  string search_trans(t)       // Searching the transition t in the BA. If ∃t' ∈ BA s.t. src(t) = src(t')
  and  dst(t) = dst(t'), then t already exists in the BA and return t'. Otherwise, return ∅.
 }
```

Algorithm 2

Table 2: LBTT parameters used for formulae generation.

Translators	Benchmark 1			Benchmark 2		
	States	Trans.	Time (s)	States	Trans.	Time (s)
LTL3BA	3938	8816	21.25	8765	25251	42.62
LTL3BA-M	3527	7611	18.57	7629	20795	35.53
LTL3BA-S	3650	7697	18.91	7586	19703	34.11
LTL3BA-M-S	3389	7002	17.32	6973	17602	30.72
Spot-BA-any-high	3442	7342	17.98	7621	20461	35.11
Spot-BA-deterministic-high	3091	6447	15.89	6306	15689	27.49
Spot-BA-small-high	2933	5863	14.66	5972	14193	25.20
ltltoba	2886	5437	14.38	5721	13817	23.32
	Benchmark 3			Benchmark 4		
	States	Trans.	Time (s)	States	Trans.	Time (s)
LTL3BA	6474	23526	50.00	18625	103332	152.44
LTL3BA-M	5772	21134	44.84	15722	79481	119.00
LTL3BA-S	6143	21382	45.88	16855	83760	125.77
LTL3BA-M-S	5561	19442	41.67	14582	67669	102.82
Spot-BA-any-high	5730	20850	44.30	17008	90438	134.31
Spot-BA-deterministic-high	5289	18531	39.70	15460	76400	124.82
Spot-BA-small-high	4927	16468	35.66	13452	59704	91.45
ltltoba	4893	15635	34.38	11708	49382	83.34

calculated according to the calculation using $(**)$-$(***)$ of Algorithm 1. If the new states do not exist in the BA, they will be added to BA. The function *add_state*( ) tells us how to add a state to the BA. Then, the transitions are added to the BA by applying the on-the-fly degeneralization algorithm (Algorithm 3, lines (24)–(30)). If the $\mathcal{S}(q_i)$ has been reset to $\emptyset$, the state q_i is the final state. The details of the main algorithm are shown in Algorithm 3.

4.3. Preliminary Analysis on Complexity. The most time-consuming component in ltltoba is the process of calculating the *cover* of the formulae. In our work, we remove the redundant states and transitions in each step of expansion. The size of the resulting automata is smaller. Representing the cover by OBDDs contributes to merging equivalent formulae. The on-the-fly degeneralization algorithm contributes to reducing the running time. Therefore, the operation of the algorithm is more efficient and the running time becomes faster. The worst time complexity of the expansion algorithm is $\mathcal{O}(2^n)$ (where n is the number of elements in f), because we have to travel all nodes in worst case. The average time complexity of our algorithm is much lower than $\mathcal{O}(2^n)$.

5. Experimental Results

The main algorithm, which is described in Section 4, is an implementation in C++ by applying CUDD library. This algorithm implements translating an LTL formula to a BA

```
(1)  Input: an LTL formula f;
(2)  Output: a BA corresponding to f;
(3)
(4)  procedure LTL_to_BA_translation
(5)  NodeSet ← {f | ∅};
(6)  BA.initialize_state(f | ∅);
(7)  while NodeSet ≠ ∅ do
(8)    let q ∈ NodeSet;
(9)    NodeSet ← NodeSet \ {q};
(10)  φ ← apply_tableau_rules(q);
(11)  for each p at the top level in φ do
(12)    φ ← (p ∧ φ[{p}]) ∨ (¬p ∧ φ[{¬p}]);
(13)    simplify(φ);
(14)  end for
(15)  DisjunctSet ← the DNF of φ;
(16)  for each d ∈ DisjunctSet do
(17)    let d ← l(t) ∧ X(q');
(18)    S(t) ← calculate using (**);
(19)    S(q') ← calculate using (***);
(20)    if BA.search_state(q') = ∅ then
(21)      BA.add_state(q' | S(q'));
(22)      NodeSet ← NodeSet ∪ {q' | S(q')};
(23)    end if
(24)    if (t' ← BA.search_trans(t)) = ∅ then
(25)      BA.add_trans(q, l(t), q', S(t))
(26)    else
(27)      let t' ← (q, l(t'), q', S(t'));
(28)      l(t') ← l(t') ∨ l(t);
(29)      S(t') ← S(t') ∪ S(t);
(30)    end if
(31)  end for
(32) end while
(33) return BA;
(34) End procedure
```

Algorithm 3: The main algorithm of ltltoba.

and removing redundant state or transition of the resulting automata. On-the-fly degeneralization is used in this algorithm. It can be used as kernel of other phases.

In this section, we extensively tested ltltoba, in comparison with the two state-of-the-art tools, LTL3BA v1.0.2 and Spot v1.1.4. We consider the running time and the number of states and transitions of the resulting automata. For the comparison of the results, we use LBTT 1.2.1 which takes a set of translation tools for testing their running time as input. LBTT [15] is a randomized testbench tool which gives a series of randomly generated formulae to the testing algorithms.

We run all tests on a computer with processor Pentium Dual-Core CPU E5300 @2.6 GHz, 4 GB of memory. The operating system was Ubuntu 12.04 LTS. Ltltoba, Spot, and LTL3BA were compiled on this machine to achieve satisfactory results. For the purpose of tests, all programs were configured with the formula simplification enabled.

We compare those translators on four sets of random formulae generated by LBTT. Benchmark 1 and Benchmark 3 contain 300 formulae of the length 15–30 and their negation. The number of atomic propositions in Benchmark 1 is 3, and the other is 8. Benchmark 2 and Benchmark 4 contain 400 formulae of the length 25–40 and their negation. The number of atomic propositions in Benchmark 2 is 3, and the other is 8. Table 2 presents the cumulative results of translations of all formulae in the four sets. The table illustrates the gradual effect of modification of each of the translators. The automata produced by ltltoba are in sum slightly better than the automata produced by other tools.

6. Conclusion

In this paper, we presented a new approach to convert an LTL formula to a BA more efficiently. Because the size of the product automata will jump exponentially when we do the product operation, the smaller and more deterministic property automata can improve efficiency of model checking. In order to deal with the infinite path during the expansion of LTL formulae, we attach the contented situation to the state and transition in the automata. In each step of expansion, we remove the redundancy states and transitions. The on-the-fly degeneralization algorithm is presented in this paper. Compared with the standard degeneralization algorithm, the on-the-fly degeneralization algorithm is a more efficient

method. A conversion tool, ltltoba, implements these ideas. Because the resulting automata are smaller during expansion, the algorithm of ltltoba is more efficient. After an extensive empirical test, *ltltoba* can reduce the running time and the number of states and transitions to some degree.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

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

[2] T. Babiak, M. Ketnsk, V. ehk et al., "LTL to Büchi automata translation: fast and more deterministic," in *Tools and Algorithms for the Construction and Analysis of Systems*, vol. 7214, pp. 95–109, Springer, Berlin, Germany, 2012.

[3] P. Gastin and D. Oddoux, "Fast LTL to Büchi automata translation," in *Computer Aided Verification*, G. Berry, H. Comon, and A. Finkel, Eds., vol. 2102 of *Lecture Notes in Computer Science*, pp. 53–65, Springer, 2001.

[4] U. Boker, O. Kupferman, and A. Rosenberg, "Alternation removal in buchi automata," in *Automata, Languages and Programming*, pp. 76–87, Springer, 2010.

[5] A. Duret-Lutz, "Ltl translation improvements in spot," in *Proceedings of the 5th International Conference on Verification and Evaluation of Computer and Communication Systems (VECoS '11)*, pp. 72–83, British Computer Society, 2011.

[6] J. M. Couvreur, "On-the-fly verification of linear temporal logic," in *FM99, Formal Methods*, vol. 1708 of *Lecture Notes in Computer Science*, pp. 253–271, Springer, Berlin, Germany, 1999.

[7] R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper, "Simple on-the-fly automatic verification of linear temporal logic," in *Proceedings of the 15th IFIP WG6 International Symposium on Protocol Specification, Testing and Verification (IFIP '95)*, 1995.

[8] G. J. Holzmann, "The model checker SPIN," *IEEE Transactions on Software Engineering*, vol. 23, no. 5, pp. 279–295, 1997.

[9] D. Giannakopoulou and F. Lerda, "From states to transitions: improving translation of LTL formulae to Büchi automata," in *Formal Techniques for Networked and Distributed Sytems—FORTE 2002*, vol. 2529 of *Lecture Notes in Computer Science*, pp. 308–326, Springer, Berlin, Germany, 2002.

[10] T. Babiak, T. Badie, A. Duret-Lutz et al., "Compositional approach to suspension and other improvements to LTL translation," in *Model Checking Software*, pp. 81–98, Springer, Berlin, Germany, 2013.

[11] K. Chatterjee, A. Gaiser, and J. Ketnsk, "Automata with generalized Rabin pairs for probabilistic model checking and LTL synthesis," in *Computer Aided Verification*, pp. 559–575, Springer, Berlin, Germany, 2013.

[12] E. Renault, A. Duret-Lutz, F. Kordon, and D. Poitrenaud, "Three SCC-based emptiness checks for generalized Büchi automata," in *Logic for Programming, Artificial Intelligence, and Reasoning*, vol. 8312 of *Lecture Notes in Computer Science*, pp. 668–682, Springer, Berlin, Germany, 2013.

[13] A. Duret-Lutz and D. Poitrenaud, "SPOT: an extensible model checking library using transition-based generalized Büchi automata," in *Proceedingsof the IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, 2004. (MASCOTS '04)*, pp. 76–83, IEEE, 2004.

[14] R. E. Bryant, "Graph-based algorithms for boolean funct ion manipulation," *IEEE Transactions on Computers*, vol. C-35, no. 8, pp. 677–691, 1986.

[15] H. Tauriainen and K. Heljanko, "Testing LTL formula translation into Büchi automata," *International Journal on Software Tools for Technology Transfer*, vol. 4, no. 1, pp. 57–70, 2002.