Conversion Algorithm of Linear-Time Temporal Logic to Büchi Automata

Laixiang Shan[a,c], Zheng Qin[b,c], Shengnan Li[a,c], Renwei Zhang[b], Xiao Yang[b]
[a] The Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China
Email: {lx.shan6, xiaoling4329}@gmail.com
[b] School of Software, Tsinghua University, Beijing 100084, China
Email: edward.gaist@gmail.com, yangxiao356@sohu.com
[c] Tsinghua National Laboratory for Information Science and Technology, Beijing 100084, China
Email: qingzh@mail.tsinghua.edu.cn

Abstract— **The size of Büchi Automata(BA) is a key factor during converting Linear-Time Temporal Logic(LTL) formulae to BA in model checking. Most algorithms for generating BA from LTL formulas involved intermediate automata, degeneralization algorithm and simplification of the formulas, but size of BA and time of converting can be reduced further. In this paper, we present an improved Tableau-based algorithm, which converts LTL formulae to Transition-based Büchi Automata(TBA) more efficiently. The algorithm is geared towards being used in model checking in on-the-fly fashion. By attaching the satisfy information of $\cup$-formula on states and transitions, we can decide whether the sequences of the BA are acceptable by using only one acceptance condition set, not multiple ones. Binary Decision Diagrams(BDD) is used to describe BA and simplify formulae. A better data structure, syntax Directed Acyclic Graph(DAG), is adopted in the algorithm. The size of product BA and computational complexity can be reduced significantly by using on-the-fly degeneralization. The algorithm can expand the state nodes while detecting the validity of nodes, removing the invalid nodes and combining equivalent states and transitions. Compared with other recent conversion tools, the algorithm proposed in this paper runs faster, with the smaller size of BA.**

Index Terms— **Büchi Automata, Linear-Time Temporal Logic, Binary Decision Diagrams, Transition-based Büchi Automata**

I. Introduction

MODEL Checking [1] is a formal analysis method that has been developed to automatically validate functional properties for software or hardware systems. The properties of system are usually specified using Linear-Time Temporal Logic or using Büchi Automata. The common method of model checking usually consists of the following steps [2]:

1) Converting system model M into BA A_M whose language $L(A_M)$ is the set of all possible execution sequences of M.
2) Transforming the negation of a given temporal property φ to a BA $A_{\neg\varphi}$ whose language $L(A_{\neg\varphi})$ is the set of all possible execution sequences that would invalidate φ.
3) Synchronized product of A_M and $A_{\neg\varphi}$. It constructs an automaton $A_M \otimes A_{\neg\varphi}$ whose language is $L(A_M) \cap L(A_{\neg\varphi})$, that is, the set of executions of the model M that would invalidate the temporal property φ.
4) Emptiness check of the product automata. This operation would tell us whether $A_M \otimes A_{\neg\varphi}$ accepts an infinite word, and can return such a counter-example if it does not. The model M verifies φ iff $L(A_M \otimes A_{\neg\varphi}) = \emptyset$ [2].

The number of states of the BA corresponding to a given LTL formula may increase exponentially in the worst case which is not expected in practice. It is very critical to make BA contain as few states and transitions as possible, because the algorithms for the emptiness check are linear to the size of BA used for verification.

In this paper, an improved Tableau-based algorithm which can generate TBA after expanding formula recursively is presented. By attaching the satisfy information of $\cup$-formula on states and transitions, We can decide whether the runs of BA are acceptable according to only one acceptance condition set. The paper focus on how to improve the conversion efficiency, so binary decision diagrams presentation and on-the-fly degeneralization are adopted in the algorithm. In order to gain significant reduction both on the size of product automata and on the computational complexity, a new algorithm which results in a vast improvement on the efficiency, especially when handing formulas containing a large amount of $G\cup-$formulae or $GF-$ formulae, is presented. Moveover, the BA is stored in quite compact form, no larger than that generated by some previous tools [3]. By comparative testing with other latest conversion tools, it is proved that the algorithm runs faster, with fewer states and transitions of the automaton.

The rest of the paper is organized as follows. In section II, we describe the related works about transform algorithm from LTL to BA. Then, we provide background information in Section III. The basic idea of the algorithm and a detailed description are presented in Section IV.

Manuscript received January 10, 2013; revised August 7, 2013; accepted September 6, 2013. © 2005 IEEE.
This work was supported in part by the Specialized Research Fund for the Doctoral Program of Higher Education (GrantNo.20120002130007), and National S&T Major Program (GrantNo.9140A1550212 JW01047).

Proof of correctness of the algorithm are introduced in Section V. In section VI, a comparison between our method and previous work is presented. Finally, Section VII closes the paper with conclusions.

II. Related Works

There are many methods of translating LTL formula to BA in the early research stage, but most of them were designed in concern of mathematic soundness but not of efficiency. [4] translated a given LTL formula into a very weak alternating automaton(VWAA) with a co-Büchi accepting condition. VWAA is then translated into a TGBA. Finally, TGBA is translated (degeneralized) into a BA. It can complete the conversion in linear time using VWAA as intermediate automaton, but the worst-case time complexity of the alternation removal is $O(n2^n)$(which is the same magnitude as alternation removal of Tableau-based algorithm) [5]. [6] proposed a method which can be used to simplify every boolean formula by using BDD. Similarly converting the BDD into an irredundant sum of products helps to reduce the number of outgoing arcs of each node. [7] proposed a classic algorithm GPVW which resorted to Generalized Büchi automaton(GBA). It can translate the property formula into a GBA, which can be then transformed into a simple BA, by using a counter mechanism proposed in [1]. GPVW algorithm is a tableau-based rule translation method in on-the-fly fashion. In the detection process, it can generate automata according to the demand in order to avoid invalid time and space consumption. GPVW algorithm has been applied in SPIN [8]. [9], [10] presented a GBA simplification method which use the table-filling algorithm of DFA. [11], [12] improved simplification method of GBA by using Fair (Bi)Simulation or Delayed (Bi)Simulation. [13] focused on identifying relationship contained by formula to avoid redundant computation. This algorithm can reduce the number of states by using the extended equivalence relation. [14] can reduce the number of states of the product of BA by considering transition-based Generalized Büchi Automata(TGBA) as an intermediate conversion automata. [15], [16] presented many simplification rules of LTL formula. It can minimize the number of temporal operators before converting the formula to GBA. [17] presented a more determinate algorithm which can be used to represent the transition. This algorithm has been used in SPOT already [2].In fact, the algorithm of traditional Tableau-based rule could generate a weak and terminal BA from a given LTL formula, which have identical topologies with their generalized counterparts, but can not produce general ones [3].

III. Preliminaries

Linear-time temporal logic is a modal temporal logic with modalities referring to time, it is a formal method of describing system constraints. In order to express a possible future situation, LTL formula describes the word on a set of 2^{AP}, namely, the infinite sequence of the set of atomic propositions(AP). Therefore, one can encode formula about the future of paths, e.g., a condition will eventually be true, a condition will be true until another fact becomes true, etc.

Definition 1 (Syntax of LTL) LTL is built up from a finite set of propositional variables AP, the logical operators $\neg$, $\wedge$ and $\vee$, and the temporal modal operators X (Next), $\cup$(Until), R(Release), F(Eventually) and G(Always). Formally, the set of LTL formulas over AP is inductively defined as follows:

- $p, \top$ and $\bot$ is LTL formula where $p \in AP$ and $\top$, $\bot$ is proposition constants.
- $\neg\varphi$, $\varphi \wedge \psi$, $\varphi \vee \psi$, Xφ, Fp, Gp, $\varphi \cup \psi$ and $\varphi R \psi$ is LTL formula, if φ and ψ is LTL formula respectively.

For an infinite word $\sigma = \sigma[0]\sigma[1]\sigma[2]... \in (2^{AP})^\omega$ over the alphabet 2^{AP}, $\sigma[i]$ denotes its ith letter, and $\sigma_i = \sigma[i]\sigma[i+1]\sigma[i+2]...$ denotes its suffix starting at letter i. R, F, G formula can be translated to $\cup$ formula by identical equation defined as follow:
- $\varphi R \psi \equiv \neg(\neg\varphi \cup \neg\psi)$
- $F\varphi \equiv \top \cup \varphi$
- $G\varphi \equiv \bot R \varphi \equiv \neg F(\neg\varphi) \equiv \neg(\top \cup \neg\varphi))$

Definition 2 (Semantics of LTL) An LTL formula can be satisfied by an infinite sequence of truth evaluations of variables in AP. These sequences can be viewed as a word on a path of a Kripke structure (an ω-word over alphabet 2^{AP}). Let $\sigma_i = \sigma[i]\sigma[i+1]\sigma[i+2]...$ be such an ω-word. Formally, the satisfaction relation between a word and an LTL formula is defined as follows:

- $\sigma \models \top$
- $\sigma \models p$ iff $p \in \sigma[0]$, for $p \in AP$
- $\sigma \models \neg\varphi$ iff $\neg(\sigma \models \varphi)$
- $\sigma \models \varphi \wedge \psi$ iff $\sigma \models \psi$ and $\sigma \models \psi$
- $\sigma \models \varphi \vee \psi$ iff $\sigma \models \psi$ or $\sigma \models \psi$
- $\sigma \models X\varphi$ iff $\sigma_1 \models \varphi$
- $\sigma \models F\varphi$ iff $\exists i \geq 0, \sigma_i \models \varphi$
- $\sigma \models G\varphi$ iff $\forall i \geq 0, \sigma_i \models \varphi$
- $\sigma \models \varphi \cup \psi$ iff $\exists j \geq 0, \sigma_j \models \psi$ and $\forall 0 \leq i < j, \sigma_i \models \varphi$

Remark 1. Every LTL formula can be rewritten as an equivalent LTL formula in negation normal form if $\neg$ unary operator occurs only immediately above atomic propositions, and $\neg, \wedge, \vee$ are the only allowed Boolean connectives. In this paper, we will consider only such formulas.

Let AP be a set of atomic propositions. We defined 2^{AP} as the set of subsets of AP and $2^{2^{AP}}$ as the set of propositional formulas induced by atomic propositions.
Definition 3 (Büchi Automata) Büchi Automata is a ω-Automata. It is a tuple B=$\langle Q, AP, \delta, I, F\rangle$, where
- Q is a finite set of states,
- AP is a set of atomic proposition,

- $\delta \subseteq Q \times 2^{AP} \times Q$ is a transition function,
- $I \in Q$ is the initial state,
- $F \subseteq Q$ is a set of accepting states.

A run ρ of B over an infinite word $\sigma = \sigma[0]\sigma[1]\sigma[2]\ldots \in (2^{AP})^{\omega}$ is a sequence $\rho = (q_0, l_0, q_1)(q_1, l_1, q_2)(q_2, l_2, q_3)\ldots \in \delta^{\omega}$, where $q_0 \in I$ is an initial state and $q_{i+1} \in \delta(q_i, \sigma[i])$ for all $i \geq 0$. The run ρ is accepted by B, if the sequence $q_0 q_1 q_2 \ldots$ visits infinitely many acceptance states ($\forall i \geq 0, \exists j \geq i\ q_j \in F$). A infinite word σ is called accepted by B if some run of B over σ is accepted. We denote by $L(B)$ the language accepted by B, i.e. the set of all words over AP accepted by B.

Definition 4 (Transition-based Büchi Generalized Automata) TGBA is a Büchi automata in which multiple acceptance conditions are carried by the transitions. It can be defined as a tuple $T = \langle Q, AP, \delta, I, F \rangle$, where
- Q is a finite set of states,
- AP is a set of atomic proposition,
- $\delta \subseteq Q \times 2^{2^{AP}} \times 2^{F} \times Q$ is a transition relation in which each transition is labeled by a Boolean formula and a set of acceptance conditions,
- $I \subseteq Q$ is a set initial states,
- $F = \{f_1, f_2, ...f_m\}$ is a finite set of elements called accepting conditions, where $f_j \subseteq Q \times 2^{AP} \times Q$ are sets of accepting transitions.

A run ρ of TGBA T over an infinite word $\sigma = \sigma[0]\sigma[1]\sigma[2]\ldots \in (2^{AP})^{\omega}$ is a sequence of states $\rho = (q_0, l_0, F_0, q_1)(q_1, l_1, F_1, q_2)(q_2, l_2, F_2, q_3)\ldots \in \delta^{\omega}$, where $q_0 \in I$ is an initial state and, for each $i \geq 0$, there exists $\alpha \in 2^{AP}$ such that $\sigma[i] \in \alpha$ and $(\alpha, q_{i+1}) \in \delta(q_i)$. A run ρ is accepted if for each $1 \leq j \leq m$ it uses infinitely many transitions from f_j, namely, $\forall i \geq 0, \exists j \geq i, \forall f \in F, f \in F_j$. A word σ is accepted if there is an accepted run over σ. We denote by $L(T)$ the language accepted by T, i.e. the set of all words over 2^{AP} accepted by an automata T.

IV. OVERVIEW OF THE ALGORITHM

In this section, we proposed an algorithm based on an improved Tableau rule construction, which generates a TBA accepting the language $L(\varphi)$ according to a given LTL formula φ. Every LTL formula φ has a corresponding language $L(\varphi)$ which can be denoted by a word $\sigma = \sigma[0]\sigma[1]\sigma[2]... \in (2^{AP})^{\omega}$. BA generated by φ will accept $L(\varphi)$. The ω-word constructed by linking the state and its successor is X-sequence generated by expanding a given LTL formula φ in accordance with the Tableau-rule, i.e., $\varphi \cup \psi = \varphi \wedge X(\varphi \cup \psi) = \varphi \wedge X\varphi \wedge XX(\varphi \cup \psi) \ldots$. This is the way the Tableau-rule works. Tableau-rule is listed in Table.1.

Table.1. Tebleau Rule

μ	$subf1(\mu)$	$next1(\mu)$	$subf2(\mu)$
$\varphi \wedge \psi$	φ, ψ	$\emptyset$	$\emptyset$
$\varphi \vee \psi$	φ	$\emptyset$	ψ
$\varphi \cup \psi$	φ	$\varphi \cup \psi$	ψ
$\varphi R \psi$	ψ	$\varphi R \psi$	$\varphi\psi$
$X\varphi$	$\emptyset$	φ	$\emptyset$

A. LTL to TBA Translation Algorithm

LTL to TBA transliation algorithm(LTTB) can convert a given LTL formula φ to TBA in accordance with φ. The idea of this new algorithm is very clear: we manage to record the satisfaction information on every state and transition during expanding the states. In order to ensure algorithm expanding, we define the variables as follows:
- *new:* the set of the unprocessed formula.
- *src:* source node, namely the set of the processed formula.
- *des:* destination node, namely the set of the successor nodes of src.
- *label:* the set of the atomic propositions using to label transition.
- *P: Promise*, accepting condition which the current transition must satisfy finally.
- *curr:* the set of LTL formula to being processed.
- *p:* Atomic proposition.
- φ*:* the input LTL formula.
- *all_pro:* the variable which record all accepting condition.

In order to describe the automata, we define the TBA class(Abbreviated as BA). Property and function of the TBA class are defined as follows:
- $BA.new()$: Generate a new BA class.
- $BA.initial()$: initialize the BA class, the initial node must save the input LTL formula.
- $BA.add\_trans()$: add a transition to BA.
- $BA.state$: The state which already exists in the BA.
- $BA.trans$: The transition which already exists in the BA.
- $BA.accept\_cond$: the set of the acceptance condition of TBA.

Remark 2. Let φ be a set of formulas. φ can be convert to a equivalent BA. $\Delta(\varphi) = \prod_{f \in F} r_f$ can be expanded to the form:

$$\prod_{f \in F} r_f = \sum_{src, P, des \in \tau} (src \wedge \prod_{g \in P} P_g \wedge \prod_{h \in des} r(src, h))$$

with $\tau \subseteq 2^{2^{AP}} \vee \{\{g \cup h \in sub(f)\} \vee \{gRh \in sub(f)\} \vee \{g \in sub(f) : Xg \in sub(f)\}$.

The algorithm start with an initial state q_0 accordance with the input LTL formula φ(Algorithm 1. line 2). Then, we can calculate the expansion recursively(Algorithm 1. line 6-9), using the Depth-First Search algorithm and Tableau-based rule for each node which has not been processed(Algorithm 2). Generally, each element

Algorithm 1 The main program algorithm of LTTB

```
1: procedure $LTTB$
2: $new \Leftarrow \varphi$
3: $all\_pro \Leftarrow \emptyset$
4: $\alpha = BA.new()$
5: $\alpha.initial(\varphi)$
6: while (new$\neq \emptyset$) do
7:     $src \Leftarrow curr \Leftarrow new$
8:     $new \Leftarrow new \setminus curr$
9:     $expand(src, des \Leftarrow label \Leftarrow P \Leftarrow \emptyset, curr)$
10: end while
11: for each t in $a.trans()$ do
12:     $t.accept\_cond \Leftarrow all\_pro \setminus t.P$
13: end for
14: End LTTB
```

$c_i \in expand(q)$ should generate a successor state q'. If q' is valid and a transition $t = (q, l(t), q')$ leading to q' is not redundant, we will add the translation into the automata. Finally, we can complete the TBA by adding every transition $t = (q, l(t), q')$ to the acceptance condition F such that $P = \emptyset$.

B. ON-THE-FLY DEGENERALIZATION SIMPLIFICATION

For the terminal BA and weak BA, the algorithm of LTTB generated almost identical graph as the one that built by traditional tableau procedures. This graph is quite compact, but this algorithm would generate correct but not optimal graph when it transforms the general automata. A large portion of states and transitions are redundant in some case. So we can remove the redundant states and transitions by using a fixed order in which the automata checks the satisfy information of the eventualities in the range of G operators.The accepting conditions marked on each transition are stored as BDD and the order used by the degeneralization algorithm is related to the order in which the corresponding BDD variables were declared. This order is in turn related to the order in which the LTL formula was recursively traversed by the transition algorithm. This reversed order corresponds to expecting the acceptance conditions associated to a formula φ before expecting the accepting condition associated to subformula $GF(a \vee F(b \vee Xc))$ there will be acceptance conditions to $a \vee F(b \vee Xc)$, $b \vee Xc$ and c, they should be expected in this order during the degeneralization. In addition to simplification on states, the algorithm can recognizes and removes redundant transitions in the process of the construction. This procedure is presented in Algorithm 3, and this algorithm should be put after Algorithm 2.line21.

C. PRELIMINARY ANALYSIS ON COMPLEXITY

The most time-consuming component in LTTB algorithm is the expanding computation. The worst time complexity of expanding algorithm is $O(2^n)$(where $n = |\varphi|$) because we have to traversal all nodes in worst case. Meanwhile, it takes $O(n)$ to complete a recursive computation in LTTB algorithm(Algorithm 1. line 6-9). Then the worst time complexity of LTTB algorithm is $O(n2^n)$. Although exponential blow-up is unavoidable, we can reduce the computation via BDD and on-the-fly simplifications. During expanding algorithm, BDD can be used to optimize specialized algorithm for calculating irredundant sum-of-products, because it is suitable to simplify formulas and remove redundant nodes. The on-the-fly simplifications will not increase too much computational complexity, but it produces TBAs of the same size as their generalized counterpart with an upper bound of $O(2^n)$ states [3].

Algorithm 2 The expansion algorithm of LTTB

```
1: procedure expand(src,des,label,P,curr)
2: if $(curr \neq \emptyset)$ then
3:     $let \mu \in curr$
4:     $curr = curr \setminus \mu$
5:     case $\mu$ of
6:         $\mu = p\ or\ \neg p\ or\ \mu = \top\ or\ \bot \Rightarrow$
7:             if $\mu = \bot\ or\ Neg(\mu) \in \alpha.state$ then
8:                 return
9:             else
10:                $label \Leftarrow label \vee curr$
11:            End if
12:        $\mu = \varphi \cup \psi\ or\ \varphi \vee \psi\ or\ \varphi\ R\ \psi\ \Rightarrow$
13:            if $\mu = \varphi \cup \psi$ then
14:                $P \Leftarrow P \vee \{\varphi \cup \psi\}$
15:            else
16:                $P \Leftarrow \emptyset$
17:                expand$(src, des \vee next1(\mu), \emptyset, P, curr \vee subf1(\mu))$
18:            expand$(src, des, \emptyset, \emptyset, curr \vee subf2(\mu))$
19:        $\mu = \varphi \wedge \psi\ or\ X\varphi \Rightarrow$
20:                expand$(src, des \vee next1(\mu), \emptyset, \emptyset, curr \vee subf1(\mu))$
21:        End case
22:    if $dest \notin a.state$ then
23:        $new \Leftarrow new \vee dest$
24:    end if
25: end if
26: return
27: End expand
```

Algorithm 3 merging transition algorithm of LTTB

```
1: procedure Add_trans$(t = (src, des, label, P))$
2: if $\exists t' \in \alpha.trans\ with\ t'.src = src\ and\ t'.dest = dest\ and\ t'.P = P$ then
3:     $trans.label \Leftarrow trans.label \vee label$
4: else
5:     $a.add\_trans(src, dest, label, P)$
6:     $all\_pro \Leftarrow all\_pro \vee promise$
7: end if
8: End procedure
```

V. Proof of Correctness

In this section, the proof of correctness will be sketched. Let $\xi = x_0x_1x_2\ldots \in (2^{\Sigma})^{\omega}$ be an infinite word accepted by BA, ξ_i denotes the suffix of the sequence ξ, i.e., $x_ix_{i+1}x_{i+2}\ldots$. The main theorem is the following:

Theorem 4.1 Let φ be an LTL formula. Let BA(φ) be the transition Büchi automaton constructed for φ. Then BA(φ) accepts exactly the infinite words ξ over the alphabet 2^{AP} that satisfy φ.
Proof. The two directions are proved in Lemma 4.2 and Lemma 4.3 below.

Lemma 4.2 Let ξ be an infinite words accepted by BA(φ). Then $\xi \models \varphi$.
Proof. By definition, if there exists an infinite path $\sigma = (q_0, l_0, F_0, q_1)(q_1, l_1, F_1, q_2)(q_2, l_2, F_2, q_3)\ldots$ in BA(φ), we considered the boolean variables $r[i]_\mu$ for each LTL formula μ, and boolean variable $P[i]_{\mu\cup\eta}$ for each accepting condition $\mu \cup \eta$. Let the sets $r[i]_{X\mu} = [\mu \in q_{i+1}]$, $r[i]_p = [p \in x_i]$ for $p \in AP$, $P[i]_{\mu\cup\eta} = [\mu \cup \eta \in f_i]$.
- 1. If p is an atomic proposition and $r[i]_p$ is true, then $p \in x_i$ and ξ_i satisfies p.
- 2. If $r[i]_{\mu\wedge\eta}$ is true, then $r[i]_{\mu\wedge\eta} = r[i]_\mu \wedge r[i]_\eta = true$ and ξ_i satisfies μ and η.
- 3. If $r[i]_{\mu\vee\eta}$ is true, then $r[i]_{\mu\vee\eta} = r[i]_\mu \vee r[i]_\eta = true$ and ξ_i satisfies μ or η.
- 4. If $r[i]_{X\mu}$ is true, then $r[i]_{X\mu} = [\mu \in q_{i+1}]$ is true and ξ_{i+1} satisfies μ, namely ξ_i satisfies μ.
- 5. If $r[i]_{\mu\cup\eta}$ is true, then $r[i]_{\mu\cup\eta} = r[i]_\mu \vee (r[i]_\mu \wedge r[i]_{X(\mu\cup\eta)} \wedge P[i]_{\mu\cup\eta}) = true$. If $r[i]_\eta = true$ then ξ_i satisfies η and $\mu \cup \eta$. If $r[i]_\mu = true$ and $r[i]_{X(\mu\cup\eta)} = true$, by definition $\mu \cup \eta \in q_{i+1}$ and then $r[i+1]_{\mu\cup\eta} = true$ and ξ_i satisfies μ. In the latter case, one can apply the same deduction for $j > i$ until $r[i]_\eta = true$. This procedure will eventually stop when $P[i]_{\mu\cup\eta}$ is false.
- 6. If $r[i]_{\mu R\eta}$ is true, then $r[i]_{\mu R\eta} = (r[i]_\mu \wedge r[i]_\eta) \vee (r[i]_\eta \wedge r[i]_{X(\mu R\eta)}) = true$. If $r[i]_\mu = r[i]_\eta = true$, then ξ_i satisfies μ, η and $\mu R\eta$. If $r[i]_{X(\mu R\eta)} = true$ and $r[i]_\eta = true$, by definition $\mu R\eta \in q_{i+1}$ and $r[i+1]_{\mu R\eta} = true$, then ξ_i satisfies η. In the latter case, one can apply the same deduction for $j > i$ until $r[j]_\mu = true$. This procedure can stop or processed infinitely. In the both case, we deduce that ξ_i satisfies $\mu R\eta$.

Lemma 4.3 Let $\xi \models \varphi$. Then ξ is accepted by BA(φ).
Proof. If $\xi \models \varphi$, each transition (q_i, l_i, F_i, q_{i+1}) corresponds to a $true$ implicant in formula $\prod_{\eta\in q_i} r_\eta$ when setting variables: $r_{X\mu} = [\xi_{i+1} \models \mu]$, $r[i]_p = [p \in x_i]$ for $p \in AP$, $P_{\mu\cup\eta} = (\xi_i \models \mu \cup \eta) \wedge \neg(\xi_i \models \eta)$. Any accepting condition $\mu \cup \eta$ appears infinitely often in the path. Otherwise there exists an $i > 0$ and a formula $\mu \cup \eta$, such that $\forall j > i$, ξ_i satisfies $\neg\eta \wedge (\mu \cup \eta)$. Then ξ is accepted by BA(φ).

VI. Performance Evaluation

In this section, we compared our algorithm with LTL3BA v1.0.2 and SPOT v1.1.1. Considering the speed of an LTL formula to Büchi automata translation, LTL3BA and SPOT are two leading tools. Both of these tools are widely used for translating LTL formulas to Büchi automata. LTL3BA is constantly outperforming many other algorithms such as GPVW [7], LTL2AUT [13], and Wring [15]. Although it is known that transition-based automata are often more compact than their state-based counterparts for the same formulas, the difference in sizes between minimal automata of both types is usually quite small. LTL3ba V1.0.2 released on December 13, 2012, is the latest version$(http : //sourceforge.net/projects/ltl3ba/?source = dlp)$.
SPOT is a C++ library offering model checking bricks that can be combined and interfaced with third party tools to build a model checker [2]. It relies on TGBA and does not need to degeneralize these automata to check their emptiness. SPOT v1.1.1 released on May 13, 2013, is the latest version$(http : //spot.lip6.fr/wiki/GetSpot)$.
LTTB uses on-the-fly simplifications described in section III, implemented in c++ using CUDD library for BDD operation.
Here are six families of formula we will evaluated on these tools. The experimental platform configuration is as follows: Intel Core i5-3470 CPU@3.2-GHz, 4GB of memory, the operating system is Ubuntu 12.04 LTS.
The first family of formula we will experiment is scalable. For a given n we generated an LTL formula C_n that matches an infinite sequence of bits in which all the values of an n-bit counter have been concatenated [6]. E.g. $C_3 = ((a\wedge(G(a \rightarrow (X(\neg a\wedge X(\neg a\wedge Xa))))))\wedge((\neg b)\wedge X(\neg b\wedge X\neg b)) \wedge (G((a \wedge \neg b) \rightarrow (X((XXb) \wedge (((\neg a) \wedge (b \rightarrow XXXb)\wedge((\neg b) \rightarrow (XXX\neg b)))\cup a)))))\wedge(G((a\wedge b) \rightarrow (X((XX\neg b) \wedge ((b \wedge (\neg a) \wedge XXX\neg b) \cup (a \vee ((\neg a) \wedge (\neg b) \wedge (X((XXb) \wedge (((\neg a) \wedge (b \rightarrow XXXb) \wedge ((\neg b) \rightarrow XXX\neg b))\cup a)))))))))))$. For this description it should be clear that the smallest automata that can recognize C_n is a deterministic loop with $n2^n$ states and transitions. Fig.5 shows this automata for C_3. Any translator that constructs such an automata explicitly will have a runtime that is worse than exponential order of n.

Secondly, we will experiment with five families of formulae. These formulae evaluated on these tools for n ranging from 5 to 9. The number of states and transitions is recorded within 30 seconds('-' represents more than 30 seconds). The five families of formula are as follow:
${\varphi_n}^G = F(p_1\wedge F(p_2\wedge\cdots Fp_n))\wedge F(q_1\wedge F(q_2\wedge\cdots Fq_n))$.
${\varphi_n}^S = ((\cdots(p_1\ R\ p_2)\cdots)\ R\ p_{n-1})\ R\ p_n$.
${\varphi_n}^O = (p_1\cup(p_2\cup\cdots p_n)\cdots)\wedge(q_1\ R\ (q_2\ R\ \cdots q_n)\cdots)$.
${\varphi_n}^P = FGp_1 \vee \cdots \vee FGp_n$.
${\varphi_n}^C = GFp_1 \wedge \cdots \wedge GFp_n$.
Table 2 lists the comparison results of LTTB and SPOT, LTL3BA on the five families of formulae.
The minimal Büchi automata for ${\varphi_n}^G$ has $(n+1)^2$ states.

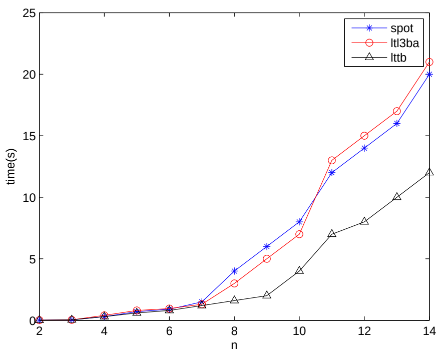

Fig.5. Runtime of these tools on LTL counter formula

All these tools were able to produce the smallest automata from $n = 5$ to 9. For ${\varphi_8}^G$, LTTB and LTL3BA produced an automaton of 81 states in about 1.5 seconds while SPOT can not produced an automata within 30 seconds. The minimal Büchi automata for ${\varphi_n}^S$ has n states. All these three tools were able to produce the smallest automata from $n = 5$ to 9 within 30 seconds.
The minimal Büchi automata for ${\varphi_n}^O$ has n^2 states. For ${\varphi_7}^O$, LTL3BA can not product an automata within 30 seconds, while SPOT can not product an automata within 30 seconds for ${\varphi_9}^O$.

The minimal Büchi automata for ${\varphi_n}^P$ has $n + 1$ states, they have to deal with an exponential number of transitions. ${\varphi_n}^P$ and ${\varphi_n}^C$ do not cause any problems on either tools. The minimal automata were produces by these tools instantaneously.

The experimental data show that, in general, LTTB is slightly faster than LTL3BA and SPOT on some formulae. With increasing parameter, LTTB outperforms LTL3BA and SPOT in the second group and the third group of experiments, while SPOT sometimes remain slower, but outperform LTL3BA finally. From table 2 we know that LTTB runs faster for ${\varphi_n}^G$ and ${\varphi_n}^O$ and produces smaller automata for ${\varphi_n}^C$.

Table 2. Size of BA produced by LTL3BA, SPOT and LTTB

LTL	LTL3BA		SPOT		LTTB		
formula	states	trans	states	trans	states	trans	time(s)
${\varphi_5}^G$	36	441	36	441	36	441	0.031
${\varphi_6}^G$	49	784	49	784	49	784	0.471
${\varphi_7}^G$	64	1294	64	1294	64	1294	0.915
${\varphi_8}^G$	81	2025	-	-	81	2025	1.128
${\varphi_9}^G$	-	-	-	-	100	3025	1.691
${\varphi_5}^S$	5	15	5	15	5	15	0.002
${\varphi_6}^S$	6	21	6	21	6	21	0.003
${\varphi_7}^S$	7	28	7	28	7	28	0.004
${\varphi_8}^S$	8	36	8	36	8	36	0.004
${\varphi_9}^S$	9	45	9	45	9	45	0.005
${\varphi_5}^O$	25	225	25	225	25	225	0.027
${\varphi_6}^O$	36	441	36	441	36	441	0.170
${\varphi_7}^O$	-	-	49	784	49	784	0.341
${\varphi_8}^O$	-	-	64	1296	64	1296	0.946
${\varphi_9}^O$	-	-	-	-	81	2025	1.733
${\varphi_5}^P$	6	11	6	11	6	11	0.001
${\varphi_6}^P$	7	13	7	13	7	13	0.002
${\varphi_7}^P$	8	15	8	15	8	15	0.002
${\varphi_8}^P$	9	17	9	17	9	17	0.003
${\varphi_9}^P$	10	19	10	19	10	19	0.003
${\varphi_5}^C$	6	26	6	26	5	10	0.020
${\varphi_6}^C$	7	34	7	34	6	12	0.036
${\varphi_7}^C$	8	43	8	43	7	14	0.070
${\varphi_8}^C$	9	53	9	53	8	16	0.136
${\varphi_9}^C$	10	64	10	64	9	18	0.359

VII. Conclusions

The size of the constructed property automata is very critical in model checking, because we should take the product of the property automata with the state automata. The size of the product automata will jump exponential when we do the product operation [18]. If the size of state automata can not be reduced, it is important that the property automata should be as small as possible. Thus, we presented an improved tableau-based algorithm to convert LTL formula to TBA more efficiently. By keeping track of the accepting conditions of all $\cup$-formula on the transitions during the construction, we can judge whether the runs of the BA is accepted by using one set of accepting conditions. By using BDD as well as on-the-fly simplification in formula presentation and cover computation, an optimized on-the-fly degeneralization is achieved. This enables the generation of compact TBAs from LTL formulas, with the time complexity, for most cases, similar to the traditional tableau-based algorithms without degeneralization and post-simplifications. Although this algorithm can not avoid exponential growth of the states, it can reduce the number of states and transitions in some degree. The experimental results show that algorithm mentioned in this paper constructed smaller Büchi Automata, with the same or reduced degree of computation complexity, on formulae in literature, by comparing with LTL3BA and SPOT.

Acknowledgment

The authors are grateful to the Professor Guiming Luo and PHD. Jian Luo for their valuable comments and suggestions to improve the presentation of this paper.

References

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

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

[3] X. Lu and G. Luo, "Direct translation of ltl formulas to büchi automata," in *Cognitive Informatics & Cognitive Computing (ICCI* CC), 2012 IEEE 11th International Conference on*. IEEE, 2012, pp. 323–328.

[4] T. Babiak, M. Křetínský, V. Řehák, and J. Strejček, "Ltl to büchi automata translation: Fast and more deterministic," in *Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2012, pp. 95–109.

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

[6] A. Duret-Lutz, "Ltl translation improvements in spot," in *Proceedings of the Fifth international conference on Verification and Evaluation of Computer and Communication Systems*. British Computer Society, 2011, pp. 72–83.

[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 Fifteenth IFIP WG6. 1 International Symposium on Protocol Specification, Testing and Verification*. IFIP, 1995.

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

[9] C. Yin and G. Luo, "Efficient translation of ltl to büchi automata," *Tsinghua Science & Technology*, vol. 14, no. 1, pp. 75–82, 2009.

[10] H. Shi, W. Ma, M. Yang, and X. Zhang, "A case study of model checking retail banking system with spin," *Journal of Computers*, vol. 7, no. 10, pp. 2503–2510, 2012.

[11] S. Juvekar and N. Piterman, "Minimizing generalized büchi automata," in *Computer Aided Verification*. Springer, 2006, pp. 45–58.

[12] I. V. Shoshmina and A. B. Belyaev, "Symbolic algorithm for generation büchi automata from ltl formulas," in *Parallel Computing Technologies*. Springer, 2011, pp. 98–109.

[13] M. Daniele, F. Giunchiglia, and M. Y. Vardi, "Improved automata generation for linear temporal logic," in *Computer Aided Verification*. Springer, 1999, pp. 249–260.

[14] D. Giannakopoulou and F. Lerda, "From states to transitions: Improving translation of ltl formulae to büchi automata," in *Formal Techniques for Networked and Distributed SytemsFORTE 2002*. Springer, 2002, pp. 308–326.

[15] F. Somenzi and R. Bloem, "Efficient büchi automata from ltl formulae," in *Computer Aided Verification*. Springer, 2000, pp. 248–263.

[16] C. Zhou and B. Sun, "Abstraction in model checking real-time temporal logic of knowledge," *Journal of Computers*, vol. 7, no. 2, pp. 362–370, 2012.

[17] R. Sebastiani and S. Tonetta, "more deterministic vs.smaller büchi automata for efficient ltl model checking," in *Correct Hardware Design and Verification Methods*. Springer, 2003, pp. 126–140.

[18] C. Fritz, "Concepts of automata construction from ltl," in *Logic for Programming, Artificial Intelligence, and Reasoning*. Springer, 2005, pp. 728–742.

Laixiang Shan received his M.S. degree in the Department of computer science and technology from University of science and technology of China, in June 2005. He is currently working towards his Ph.D. degree in Computer Science and Technology at Tsinghua University, China. His current research interest includes Model Checking, software testing and modeling Theory.

QIN Zheng was born in Hunan Province, China, in 1956. He is a pro-fessor in the School of Software, Tsinghua University, China. His major research in-terest includes software architecture, data fusion and artificial intelligence. (E-mail:qingzh@mail.tsinghua.edu.cn)

Shengnan Li received the B.S. degree in software engineering from Nankai University, China in June 2011. She is currently working towards his Ph.D. degree in Computer Science and Technology at Tsinghua University, China. Her current research interest includes wireless sensor network and target tracking.

Renwei Zhang received his master degree in software engineering from Peking University, Beijing, China in 2012. He is currently working towards his Ph.D. in Tsinghua University, Beijing, China. His research interests include software engineering, software testing and modeling Theory.

Xiao Yang received his M.S. degree in the Department of NUDT of China, in December 2011. He is currently working towards his Ph.D. degree in Software Engineering at Tsinghua University, China. His research interests include Intelligent Transportation Systems, Pedestrian Simulation and Pattern Recognition.