Fast Translation from LTL to Büchi Automata via Non-transition-based Automata

Shohei Mochizuki, Masaya Shimakawa, Shigeki Hagihara, and Naoki Yonezaki

Department of Computer Science,
Graduate School of Information Science and Engineering,
Tokyo Institute of Technology.
2-12-1-W8-67 Ookayama, Meguro-ku, Tokyo 152-8552, Japan

Abstract. In model checking, properties are typically defined in linear temporal logic (LTL) and are translated into non-deterministic Büchi automata (NBA). In this paper, we propose a new, efficient translation method that is different from those used in LTL2BA, Spot and LTL3BA. Our method produces non-transition-based generalised Büchi automata (GBA) as an intermediate object, whereas LTL2BA, Spot, and LTL3BA use transition-based generalised Büchi automata (TGBA). Our method enables fast conversion because the data structure representing the object is simpler than that used in conversions via TGBA. Furthermore, we have developed techniques to reduce the number of states, similar to techniques that have heretofore only been available for conversions via TGBA. We also propose a technique to suppress the increase in the number of states that normally occurs while GBA is converted into NBA, using characteristics of strongly connected components of the GBA. We implemented our method with these techniques and experimentally compared our method with LTL2BA, Spot, and LTL3BA, which are the fastest translators to date. Our conversion method was much faster than LTL2BA and Spot, and was competitive with LTL3BA. In addition, the number of states in the NBA resulting from our method was comparable to that produced by LTL2BA, Spot, and LTL3BA.

1 Introduction

Recently, formal methods have become essential tools for developing safety critical systems, where behavioural correctness of the systems is the main concern. For instance, model checking [11] is a method for checking whether models of systems satisfy specifications. Satisfiability checking [14] is a method for checking whether specifications are free of contradictions. Realisability checking [13,1] is a method for checking whether a program that satisfies specifications exists and includes synthesis of the program if it does [7,12]. Of these methods, linear temporal logic (LTL) is often used for describing the specifications of systems. In this case, algorithms for converting specifications written in LTL into non-deterministic Büchi automata (NBA) are commonly used. The time complexity for conversion of LTL formulae into NBA is $2^{O(n)}$, where n is the number of

S. Merz and J. Pang (Eds.): ICFEM 2014, LNCS 8829, pp. 364–379, 2014.

formulae. Especially for realisability checking, because a specification includes all the constraints of behaviour of an intended system, the size of the specification can become very large. Therefore, efficient algorithms for converting LTL formulae into NBA are strongly desirable to expand the applicable range of these checking methods.

Many translation tools for converting LTL formulae into NBA have been proposed, such as the tool implemented in the model checker SPIN [9] and several more efficient tools, including LTL2BA [8], Spot [6,5], and LTL3BA[4]. Translation methods are roughly divided into two kinds: methods for conversion via generalised Büchi automata (GBA) as an intermediate object, and methods for conversion via transition-based generalised Büchi automata (TGBA). The methods via GBA were originally the most popular. However, since 2002, the methods via TGBA have become more popular and have outpaced the methods via GBA. The efficient tools LTL2BA, Spot and LTL3BA are classified as methods of conversion via TGBA. Because TGBA can express a given accepting language by a fewer number of states than GBA, TGBA are generally smaller than GBA. This is because an acceptance condition in GBA is defined by the set of final states that are passed infinitely, while an acceptance condition in TGBA is defined by the set of transitions that are passed infinitely. On the other hand, the data structure representing GBA is simpler than that representing TGBA because the number of states is much less than the number of transitions. From this observation, it follows that if techniques for the reduction of states used for conversion via TGBA can be implemented for conversion via GBA, the methods using GBA would be expected to be extremely efficient. In this paper, we adopted the GBA conversion, imported the reduction techniques into it, implemented it and evaluated its efficiency.

Unfortunately, the techniques for reducing the number of states in TGBA, as adopted in Spot and LTL2BA, are not directly applicable to a method of conversion via GBA. Therefore, we developed comparable reduction techniques that can be applied to the conversion of GBA to NBA. This enables us to produce NBA with a comparable number of states as that resulting from TGBA conversion using reduction techniques.

For converting LTL formulae into GBA, we adopted the algorithm proposed by Aoshima et al. [2], with some modification. In its original form, this algorithm intentionally does not execute full LTL formulae, but rather executes LTL formulae without the next operator. This is to prevent introducing unintentional synchronisation by two or more different occurrences of the next operator. However, because our aim was to make it possible to convert general LTL formulae, we extended the algorithm to enable its application to LTL formulae with the next operator. In addition to the reduction techniques mentioned above for converting GBA into NBA, we also developed a technique to suppress the increase in number of states as GBA is converted into NBA, using characteristics of strongly connected components of GBA. We implemented our method using these techniques and experimentally compared our method to LTL2BA, Spot and LTL3BA, which are currently the fastest translators. Our conversion

method is faster than Spot and LTL2BA, and is competitive with LTL3BA. In addition, the number of states in the NBA from our method is comparable to that from the other methods.

The remainder of this paper is organised as follows. In Sect. 2, we give definitions of LTL and Büchi automata. In Sect. 3, we propose a new method (an extension of our previous method) to convert LTL formulae into GBA. In Sect. 4, we explain how to convert GBA into NBA. In Sect. 5, we describe our techniques for reducing the number of states in the resulting NBA. In Sect. 6, we discuss the advantages of our method over other approaches. In Sect. 7, we describe the implementation of our method and compare it to LTL2BA, Spot and LTL3BA. Finally, we present our conclusions in Sect. 8.

2 Preliminary

In this section, we introduce the syntax and semantics of LTL, NBA and GBA.

2.1 LTL

Let *Prop* be a finite set of propositions.

Definition 1 (LTL formulae). *Formulae f in LTL are inductively defined as follows:*

$$f ::= p \mid \neg f \mid f \vee f \mid f \wedge f \mid Xf \mid fUf \mid fRf,$$

where $p \in Prop$.

The notation Xf states 'f holds at the next time', while fUg represents 'f always holds until g holds'. fRg is the dual connective of fUg and represents $\neg(\neg fU\neg g)$. The notation $f \rightarrow g$, $f \leftrightarrow g$, $\top$, $\bot$, Ff and Gf are abbreviations for $\neg f \vee g$, $(\neg f \vee g) \wedge (\neg g \vee f)$, $p \vee \neg p$, $\neg\top$, $\top Uf$, and $\neg F\neg f$, respectively.

Definition 2 (Semantics). *Let Σ be 2^{Prop}, and let $u = u_0u_1, \ldots$ be an infinite sequence over Σ. Let f be an LTL formula. When a formula f holds on u, we write $u \models f$, and inductively define this relation as follows.*

- $u \models p$ *iff* $p \in u_0$
- $u \models \neg f_1$ *iff* $u \not\models f_1$
- $u \models f_1 \wedge f_2$ *iff* $u \models f_1$ *and* $u \models f_2$
- $u \models f_1 \vee f_2$ *iff* $u \models f_1$ *or* $u \models f_2$
- $u \models Xf_1$ *iff* $u_1u_2 \ldots \models f_1$
- $u \models f_1Uf_2$ *iff* $\exists k \geq 0((u_ku_{k+1} \ldots \models f_2)$ *and* $\forall i(0 \leq i < k.\ u_iu_{i+1} \ldots \models f_1))$
- $u \models f_1Rf_2$ *iff* $\forall k \geq 0((u_ku_{k+1} \ldots \models f_2)$ *or* $\exists i(0 \leq i < k.\ u_iu_{i+1} \ldots \models f_1))$

A formula is in negation normal form (nnf) if the negation symbol ($\neg$) occurs only immediately above elementary propositions. Every formula can be transformed to an equivalent formula in nnf. We call a formula f a temporal formula if f is of the form Xf_1, f_1Uf_2, or f_1Rf_2.

2.2 Automata

In this section, we introduce NBA and GBA. NBA is an automaton that accepts ω-words if there exists a corresponding run passing a final state infinitely often, which is defined as follows.

Definition 3 (Büchi automata). *Let Prop be a set of propositions. A non-deterministic Büchi automaton on an alphabet 2^{Prop} is defined by $\mathcal{A} = \langle Q, \Sigma, \delta, I, F \rangle$, where Q is a finite set of states, $\Sigma = 2^{Prop}$, $\delta \subseteq Q \times B(Prop) \times Q$ is a transition relation, $I \subseteq Q$ is a set of initial states, and $F \subseteq Q$ is a set of final states. $B(Prop)$ is a set of Boolean formulae which consist of propositions in Prop and connectives $\neg$, $\vee$, and $\wedge$. A run r of $\mathcal{A}$ on an ω-word $u = u_0 u_1 \ldots$ is an infinite sequence $q_0 q_1 \ldots$ of states, where $q_0 \in I$, $(q_i, b_i, q_{i+1}) \in \delta$, and $u_i \models b_i$ for some b_i for all $i \geq 0$. If $Inf(r) \cap F \neq \emptyset$ holds, a run r is said to be successful, where $Inf(r)$ is a set of states that occur infinitely often in r. If there is a successful run of $\mathcal{A}$ on u, we say that $\mathcal{A}$ accepts u.*

On the other hand, GBA is an automaton with multiple sets of final states (a set of sets of final states). A run is successful if, for each set of final states, it passes infinitely often some state from the set.

Definition 4 (Generalised Büchi Automata). *Let Prop be a set of propositions. A Generalised non-deterministic Büchi automaton on an alphabet 2^{Prop} is defined by $\mathcal{A} = \langle Q, \Sigma, \delta, I, \mathcal{F} \rangle$, where Q, Σ, δ and I are defined as above for NBA. $\mathcal{F} = \{F_1, \ldots, F_n\}$ is a set of sets of final states, and satisfies $F_i \subseteq Q$ for all $1 \leq i \leq n$. A run r is said to be successful if $\forall F_i (Inf(r) \cap F_i \neq \emptyset)$ holds. If there is a successful run of $\mathcal{A}$ on u, we say that $\mathcal{A}$ accepts u.*

A set of ω-words that are accepted by NBA (or GBA) $\mathcal{A}$ is called the language accepted by $\mathcal{A}$, which is represented by $L(\mathcal{A})$.

3 Converting LTL Formulae into GBA

In this section, we propose an algorithm for constructing GBA $\mathcal{A}_\varphi$ from an LTL formula φ, which satisfies $L(\mathcal{A}_\varphi) = \{u \in (2^{Prop})^\omega \mid u \models \varphi\}$. This algorithm is an extended version of a previous algorithm proposed by Aoshima et al. [2], modified to work with LTL with the next operator. Below, we explain the algorithm, and for simplicity of explanation, assume that the input LTL formulae are in nnf.

Let φ be an input LTL formula. A state of GBA consists of a subset of $cl(\varphi) \cup \{(fUg)^{unsat} \mid fUg \in cl(\varphi)\}$, which represents the constraints of the state. Here, $cl(\varphi)$ is the set of subformulae of φ. First, an initial state consists of a singleton $\{\varphi\}$ of an input formula. Next, we decompose the formulae in a state and obtain the set of successive states. We take notice if the state involves the 'until' formula fUg because its meaning has eventuality. If g holds in the state, we accept transition to the state involving no constraints on the 'until' formula. If f holds in the state, we accept transition to the state involving $(fUg)^{unsat}$. The label *unsat* represents 'eventuality (g in this case) is not satisfied'. By setting

the transition relation as stated above, if a state does not involve the labelled formula $(fUg)^{unsat}$, we can capture that g holds, (i.e., fUg holds.) If r is a run on an ω-word, such that fUg does not hold and f always holds, then the run r will stay only in states involving $(fUg)^{unsat}$. We judge the run to be successful only if the run infinitely often visits a state that does not involve $(fUg)^{unsat}$.

The procedure *Next*, used to obtain the set of transitions, is defined as follows.

Procedure 1 (*Next*) *Procedure Next takes a state $q = \{\varphi_1, \ldots, \varphi_n\}$ as input and outputs the set of transitions from q. Each transition is of the form (q, b, q'), which indicates that q' is a successive state of q by valuation satisfying b.*

1. $\Sigma := \{q\}$
2. *Repeat the following operations until Σ does not change. For every $S_i \in \Sigma$, apply one of the following, according to $f_{ij} \in S_i$.*
   (a) *if f_{ij} is of the form $f_1 \wedge f_2$, replace S_i with $(S_i - \{f_{ij}\}) \cup \{f_1, f_2\}$.*
   (b) *if f_{ij} is of the form $f_1 \vee f_2$, replace S_i with $(S_i - \{f_{ij}\}) \cup \{f_1\}$, $(S_i - \{f_{ij}\}) \cup \{f_2\}$.*
   (c) *if f_{ij} is of the form $f_1 U f_2$ or $(f_1 U f_2)^{unsat}$, replace S_i with $(S_i - \{f_{ij}\}) \cup \{f_2\}$, $(S_i - \{f_{ij}\}) \cup \{f_1, X(f_1 U f_2)^{unsat}\}$.*
   (d) *if f_{ij} is of the form $f_1 R f_2$, replace S_i with $(S_i - \{f_{ij}\}) \cup \{f_1, f_2\}$, $(S_i - \{f_{ij}\}) \cup \{f_2, X(f_1 R f_2)\}$.*
3. *Output the following δ_q.*

$$\delta_q = \{(q, \bigwedge_{l \in P(m)} l,\ \{f \mid Xf \in m\}) \quad \mid m \in \Sigma\},$$

where $P(m) = \{p \mid p \in m \wedge p \in Prop\} \cup \{\neg p \mid \neg p \in m \wedge p \in Prop\}$.

In step 2, we obtain multiple sets of formulae by decomposing a formula in a set of formulae according to the semantics of LTL. For instance, because $f_1 U f_2$ indicates that f_2 holds eventually and f_1 always holds until f_2 holds, $\{f_1 U f_2\}$ is separated into two cases $\{f_2\}$ and $\{f_1, X(f_1 U f_2)^{unsat}\}$. These cases mean "$f_2$ currently holds" and "f_1 currently holds and $f_1 U f_2$ holds at the next time", respectively. In brief, if a set of formulae is obtained by step 2, the set represents one of satisfaction of $\varphi_1 \wedge \ldots \wedge \varphi_n$ involved by q. Therefore, in step 3, atomic propositions and their negation in a set of formulae are considered a label of the transition (i.e., a condition of the transition), and a set of formulae obtained by eliminating the next operator X is considered a successive state of q.

Example 1. Let φ be $G(r \rightarrow Fs)$. We apply procedure *Next* to φ. The result of step 2 is as follows.

$$\{\{\neg r, X\varphi\}, \{s, X\varphi\}, \{X(Fs)^{unsat}, X\varphi\}\}$$

The result of step 3 is the following transitions.

$$\{(\{\varphi\}, \neg r, \{\varphi\}), (\{\varphi\}, s, \{\varphi\}), (\{\varphi\}, \top, \{(Fs)^{unsat}, \varphi\})\}$$

These transitions are depicted in Fig.1.

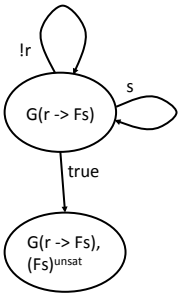

Fig. 1. Transitions generated by *Next* applied to $G(r \rightarrow Fs)$

By procedure *Next*, we can obtain the set of successive states of a state. Hence, by setting the initial state as a singleton $\{\varphi\}$ of an input formula and applying procedure *Next* iteratively, we can obtain a transition system that is part of $\mathcal{A}_\varphi$. This procedure is defined as follows.

Procedure 2 (*Construct*) ***Input:*** *φ: formula*
Output: Q, δ
Procedure: *Construct*
1: $Q = \{\{\varphi\}\}$
2: $S = \{\{\varphi\}\}$
3: $\delta = \{\}$
4: ***while*** $S \neq \emptyset$ ***do***
5: *Pick q from S and $S = S - \{q\}$*
6: δ_q *= Next(q)*
7: $\delta = \delta \cup \delta_q$
8: ***for all*** $(curstate, b, nextstate) \in \delta_q$ ***do***
9: ***if*** $nextstate \notin Q$ ***then***
10: $Q = Q \cup \{nextstate\}$
11: $S = S \cup \{nextstate\}$
12: ***end if***
13: ***end for***
14: ***end while***

Next, we define a set of sets of final states as follows. Let $\varphi_1, \ldots, \varphi_n$ be 'until' formulae, which are subformulae of an input formula φ. The set of sets of final states is $\mathcal{F} = \{F_1, \ldots, F_n\}$, where F_i is the set of states that does not include φ_i^{unsat}.

This method is an extension of our previous method [2], modified to adopt the label *unsat* for setting acceptance conditions correctly. Any successive states of a state involving $X(fUg)$ have fUg. On the other hand, any successive states of a state involving fUg have a labelled formula $(fUg)^{unsat}$ only if g does not hold in the state. With the label *unsat*, we can manage successive states of a state q involving both $X(fUg)$ and fUg. Even if fUg is contained in a successive state

q' of q, if $(fUg)^{unsat}$ is not contained in q', it indicates that fUg is satisfied in q. Without introduction of the label *unsat*, it is impossible to judge whether fUg is satisfied in q for such a case.

Example 2. The formula $GXFp$ is translated into GBA, as shown in Fig. 2 (left), by the method proposed in this section. The set of sets of final states is $\{\{q_1, q_2\}\}$. Due to the label *unsat*, we can determine the final states $\{\{q_1, q_2\}\}$ appropriately. Without introduction of *unsat*, only the transition system shown in Fig. 2 (right) can be obtained, and the appropriate set of sets of final states cannot be determined. This illustrates why it is essential to introduce the label *unsat* to know the acceptance conditions of GBA.

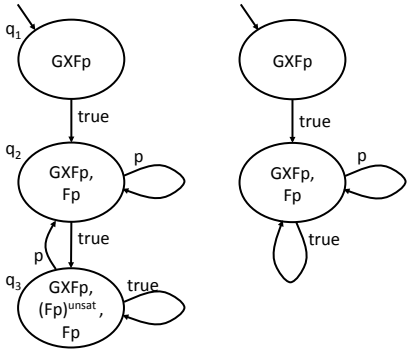

Fig. 2. GBA for $GXFp$, obtained by procedure *Construct*(left), and an incorrect transition system (right)

In the previous method proposed by Aoshima et al. in [2], binary decision diagrams (BDDs) were used to represent transitions from a state, and a BDD-based version of procedure *Next* was also used. In this current work, we have extended the previous method to permit LTL with the next operator, and we adopted the BDD-based procedure *Next*.

4 Converting GBA into NBA

An algorithm that converts TGBA into NBA was proposed for the tool LTL2BA [8]. We modified this algorithm as follows to convert GBA into NBA.

Definition 5 (Translation from GBA to NBA). *Let* $\mathcal{A} = (Q, \Sigma, \delta, I, \mathcal{F} = \{F_1, \ldots, F_k\})$ *be a GBA. The following NBA* $\mathcal{B} = (Q', \Sigma, \delta', I', F')$ *satisfies* $L(\mathcal{A}) = L(\mathcal{B})$.

- $Q' = Q \times \{1, \ldots, k+1\}$, $I' = I \times \{1\}$, $F' = Q \times \{k+1\}$

– $((s,j),a,(t,i)) \in \delta'$ *iff*

$$(s,a,t) \in \delta \wedge \begin{cases} i = next(j,t) & \text{if } j \neq k+1 \\ i = next(1,t) & \text{if } j = k+1 \end{cases},$$

$$\text{where } next(i,q) = \begin{cases} min\{j \mid i \leq j \wedge q \notin F_j\} & \text{if } \exists j \geq i\ (q \notin F_j) \\ k+1 & \text{otherwise} \end{cases}.$$

States of the resulting NBA $\mathcal{B}$ are pairs of a state in GBA $\mathcal{A}$ and an integer between 1 and $k+1$ (called a counter). This counter is important to translate GBA into NBA. Assume that there is transition (s,a,t) in $\mathcal{A}$. Let us consider the transition from (s,i) of $\mathcal{B}$. If $i \neq k+1$, $\mathcal{B}$ has transition $((s,i),a,(t,i+3))$ for the case of $t \in F_i \cap F_{i+1} \cap F_{i+2}$ and $t \notin F_{i+3}$. If $i = k+1$, $\mathcal{B}$ has transition $((s,k+1),a,(t,3))$ for the case of $t \in F_1 \cap F_2$ and $t \notin F_3$. The set of final states of $\mathcal{B}$ is the set of states in which the counter equals $k+1$. By this definition of final states of NBA, a run infinitely often passes a final state of $\mathcal{B}$ if and only if for every set of final states of $\mathcal{A}$, there is a final state such that the corresponding run infinitely often passes it.

5 Reducing States of Automata

For checking verification properties, such as satisfiability and realisability of specifications written in LTL formulae, automata manipulations such as emptiness checking, determinisation, and complementation are required. To do these kinds of manipulations of NBA efficiently, it is important to reduce the number of states of NBA, without changing the accepting languages. In our work, we adopted the formulae rewriting technique proposed in [15]. Furthermore, in this section, we propose two kinds of reduction techniques. In Sect. 5.1, we propose a technique for reducing states of NBA based on strongly connected components of GBA. In Sect. 5.2, we propose another technique for reducing states of NBA based on equivalence of states in the GBA. Generally, many reduction techniques based on simulation were proposed. This kind of techniques can be applied after the entire autotmata were constructed. The techniques we propose in Sect.5.1 and 5.2 are lightweight and can be applied in the middle of construction of the automata. This reduces time and space required for construction of automata.

5.1 Reduction of NBA Based on SCC of GBA

In the translation method proposed in Sect. 4, $k+1$ states will be copied from a state in the GBA, where k is the number of sets of final states in the GBA. However, if a state q is not included in any strongly connected components that include final states of the GBA, then it is not necessary to copy q because it is not necessary to check the acceptance condition by q.

Formally, let $\mathcal{F}$ be a set of sets of final states in GBA, and let S be a strongly connected component in GBA. We say that S is acceptable if it satisfies the following condition.

$$\begin{aligned} &(|S| > 1 \wedge \forall F \in \mathcal{F} \exists q \in S(q \in F)) \\ \vee\ &(|S| = 1 \wedge \exists q \in S(\exists a(q,a,q) \in \delta \wedge \forall F \in \mathcal{F}(q \in F))) \end{aligned}$$

If a state q is included in a maximal strongly connected component that is not acceptable, then we do not copy q when we convert GBA to NBA, and we do not include q in the set of final states of NBA.

5.2 Reduction of NBA Based on Equivalence of States in the GBA

In GBA, it is possible to identify multiple equivalent states, and to reduce the number of states in the GBA by combining these equivalent states into one state.

Formally, let $\mathcal{F}$ and δ be a set of sets of final states and a transition relation in the GBA, respectively. We say that states q_1 and q_2 are equivalent if the following two conditions hold.

$$\forall b \in B(Prop) \forall q \in Q((q_1, b, q) \in \delta \Longleftrightarrow (q_2, b, q) \in \delta) \tag{1}$$

$$\forall F \in \mathcal{F}(q_1 \in F \Longleftrightarrow q_2 \in F) \tag{2}$$

If q_1 and q_2 in the GBA satisfy both conditions (1) and (2), we can combine q_1 and q_2 into one state.

Furthermore, even if states q_1 and q_2 in the GBA satisfy condition (1) only, states (q_1, i) and (q_2, i) in the NBA converted according to Def. 5 in Sect. 4 satisfies both conditions (1) and (2) by the following theorem.

Theorem 1. *Let q_1 and q_2 be states in GBA $\mathcal{A}$, and (q_1, i) and (q_2, i) be states in NBA $\mathcal{B}$, which is obtained according to Def. 5. Then, if q_1 and q_2 satisfy condition (1), (q_1, i) and (q_2, i) satisfy condition (1) and the following (2'):*

$$(q_1, i) \in F \Longleftrightarrow (q_2, i) \in F \tag{2'}$$

where F is the set of final states in NBA $\mathcal{B}$.

Proof. It is trivial that (q_1, i) and (q_2, i) satisfy condition (2'), due to the definition of final states of $\mathcal{B}$ in Def. 5. We show that (q_1, i) and (q_2, i) satisfy condition (1). Assume that there is transition $((q_1, i), a, (s, j))$ in $\mathcal{B}$. Then, there is transition (q_1, a, s) in $\mathcal{A}$. Since q_1 and q_2 satisfy condition (1), there is transition (q_2, a, s) in $\mathcal{A}$. According to Def. 5, there exists transition $((q_2, i), a, (s, j'))$ in $\mathcal{B}$. Now, if $i \neq k + 1$ holds, then $j' = next(i, s) = j$ holds, and if $i = k + 1$ holds, then $j' = next(1, s) = j$ holds, where k is the number of sets of final states in $\mathcal{A}$. Hence, $j' = j$ holds. Therefore if there is a transition $((q_1, i), a, (s, j))$ in $\mathcal{B}$, there is also a transition $((q_2, i), a, (s, j))$ in $\mathcal{B}$. This means (q_1, i) and (q_2, i) satisfy condition (1). □

According to Theorem 1, we can obtain reduced NBA directly, without calculating large-scale NBA, by producing the reduced NBA while checking whether states in the GBA satisfy condition (1).

Example 3. We show a GBA converted from the LTL formula $GFp_1 \wedge GFp_2$ in Fig. 3 (left). The initial state is q_1, and a set of sets of final states is $\{\{q_1, q_3\}, \{q_1, q_2\}\}$. The initial state q_1 has four successive states q_1, q_2, q_3, q_4. Any two

different states $q_i, q_j \in \{q_1, q_2, q_3, q_4\}$ do not satisfy condition (2), but they always satisfy condition (1). Unfortunately, because conditions (1) and (2) are not both satisfied, we cannot reduce the GBA itself directly. However, due to the fact that condition (1) is satisfied, we can apply our reduction technique to the conversion of the GBA to an NBA, and all of the states $(q_1, i), \ldots, (q_4, i)$ in the resulting NBA can be reduced into one state (called i). The resulting NBA is shown in Fig. 3 (right).

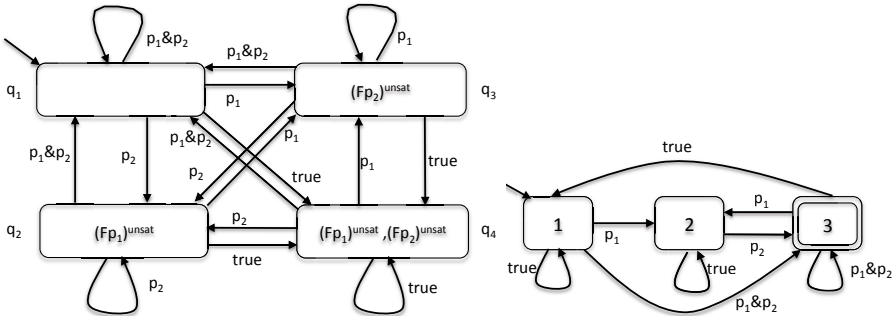

Fig. 3. GBA converted from $GFp_1 \wedge GFp_2$ (left) and a reduced NBA converted from the GBA (right)

6 Advantages of Our Method Over a TGBA-Based Approach

6.1 Features of Our Method

Translation methods are roughly divided into two kinds: those for conversion via GBA as an intermediate object, and those for conversion via TGBA. Originally, conversion via GBA was the most widely adopted method. However, since 2002, most methods are based on conversion via TGBA, (e.g., [10]). This is due to the development of efficient tools such as LTL2BA, Spot and LTL3BA (presented in 2012), all of which are based on conversion via TGBA. One advantage of the TGBA method is that TGBA is smaller than GBA for any given accepting language.

On the other hand, our method is based on conversion via GBA. Unfortunately, techniques that reduce the number of states of TGBA cannot be applied to GBA, as GBA itself cannot be reduced. Hence, we introduced a technique (Sect. 5.2) that has the same effect on the resulting NBA converted from GBA as reduction techniques have on TGBA. This technique can be applied while converting GBA into NBA, which means that GBA can be converted directly into reduced NBA. With this technique, although the number of states of GBA is larger than the number of states of TGBA, the number of states in the resulting NBA from GBA is comparable to the number in the NBA converted

from TGBA. As illustrated in Example 3, even in case of $GFp_1 \wedge \ldots \wedge GFp_n$,[1] the number of states in the resulting NBA are the same. Detailed results are presented in Sect. 7.

6.2 Our Method vs. the TGBA-Based Approach

About reduction of states As stated above, during conversion via TGBA, 'strong' reduction techniques can be applied while constructing TGBA as an intermediate object. This 'strong' reduction is done by calculating successive states of each state and checking for state equivalence. Hence, it does not result in a substantial decrease in the computing cost of constructing the TGBA. On the other hand, our reduction technique is not applied to GBA but rather to NBA. In our reduction technique, we do not calculate successive states of NBA, but check state equivalence only by the labels ((q, i): tuples of states of GBA and counter) and information in the GBA. This does result in a decrease in computing cost for constructing the NBA. Because our reduction technique has the same effect on NBA as the 'strong' reductions used in conversions via TGBA, the cost of constructing NBA by our method is expected to be no higher than the cost of constructing NBA during conversions via TGBA. For this reason, we conclude that conversion via TGBA is not always advantageous from the point of view of reduction, compared to conversion via GBA.

About acceptance conditions We have observed that in TGBA converted from LTL formulae, before the application of any reduction, there is only two cases: all the transitions into the same state are included in an acceptance condition, or no transitions into the state is included in the acceptance condition.[2] For instance, in the translation algorithm proposed in [8], if a formula Xf, where f represents eventuality (e.g. $X(fUg)$), does not occur in the input formula, then there is only two cases: all the transitions into the same state are included in an acceptance condition, or no transitions into the state is included in the acceptance condition. Therefore, with respect to efficiency of memory use, the acceptance condition should be defined by a set of states, not by a set of transitions. Hence, in our method, we treat formulae of the form $X(fUg)$ as exceptions by using the special label *unsat*, as stated in Sect. 3. This has the disadvantage of increasing the number of states. However, there are few occurrences of formulae of the form $X(fUg)$ in practical specifications. Therefore, we expect that the advantage of being able to omit redundant space in the acceptance conditions is greater than any disadvantage arising from an increase in the number of states.

Necessity of NBA It is evident that TGBA is preferable to GBA for use in model-checking directly, because the number of states of TGBA is less than the number of states of GBA or NBA. On the other hand, there are many cases in

[1] In the conversion of $GFp_1 \wedge \ldots \wedge GFp_n$, the conversion via TGBA works very well.

[2] This observation is satisfied only for TGBA before the application of any reduction techniques including ones based on equivalence of successor states.

which NBA is needed, such as for SPIN or realisability checking. In such cases, the number of states of TGBA or GBA as intermediate objects is unimportant. Therefore, we conclude that our method is valuable.

7 Evaluation

We implemented our method in C++. Our implementation is available at http://www.yonezaki.cs.titech.ac.jp/tools/. In this section, we compare our implementation to other tools. The comparison environment was OS:Ubuntu 12.04 64bit, CPU: Corei7-3820 3.60GHz, 32GB memory.

7.1 Comparison with Other Works

Although there are many tools for converting LTL into NBA, we compared our implementation to LTL2BA (version 1.1, without options), Spot (version 1.1.4, with the option `-r1`)[3] and LTL3BA (version 1.0.2, without options), because Rozier et al. in [14] showed that LTL2BA [8] and Spot [6] were the most efficient tools at that time, and LTL3BA, presented in 2012, is also known as one of the most efficient tools. These tools are based on conversion via TGBA.

We measured the times for conversion of several LTL formulae. If the tools could not output the results within 300 s, we aborted the execution; this is denoted in the results by na. For our implementation, we measured the conversion times with and without the reduction techniques proposed in Sect. 5.

First, we generated 100 random formulae of equal size (50 characters), following the method of [14], and measured the sum of the conversion times to NBA, as well as the sum of the number of states in the resulting NBA. These results are shown in Table 1.

Table 1. The sum of the number of states and the sum of conversion times (s) for 100 random formulae

Our implementation				Spot		LTL2BA		LTL3BA	
with reduction		without reduction		# of		# of		# of	
# of states	time	# of states	time	states	time	states	time	states	time
37652	2.58	83492	3.10	29308	14.00	65903	64.15	46303	24.00

Next, we measured the conversion times and number of states for the specification of n-floor elevator systems [3]. This is a large-scale specification, with $3n + 6$ propositions and $6n - 1$ temporal operators. The size of the specification is $O(n^3)$. These results are shown in Table 2.

Finally, we measured the conversion times and the number of states for the following four kinds of LTL formulae, which were used as benchmarks in [5].

- $E_1(n) : \bigwedge_{1 \leq i \leq n} Fp_i$

[3] In Spot, the option `-r1` means that formula reduction using basic rewriting is allowed.

Table 2. The number of states and conversion times (s) for specifications of n-floor elevator systems

	Our implementation				Spot		LTL2BA		LTL3BA	
	with reduction		without reduction		# of		# of		# of	
n	# of states	time	# of states	time	states	time	states	time	states	time
2	24	0.01	24	0.01	23	0.05	23	0.04	23	0.01
3	182	0.02	182	0.02	170	0.15	224	5.95	182	0.06
4	1438	0.16	1757	0.17	1333	2.44	na	na	1385	1.83
5	10403	2.46	16660	3.01	9585	53.96	na	na	9524	118.33
6	69685	43.23	145786	90.62	na	na	na	na	na	na

- $E_2(n) : F(p_1 \wedge F(p_2 \wedge \ldots F(p_{n-1} \wedge Fp_n)\ldots)) \wedge F(q_1 \wedge F(q_2 \wedge \ldots F(q_{n-1} \wedge Fq_n)\ldots))$
- $U(n) : (\ldots(((p_1 U p_2) U p_3) U p_4)\ldots) U p_n$
- $C(n) : \bigwedge_{1 \leq i \leq n} GFp_i$

These results are shown in Table 3.[4] The formulae $C(n)$ are known to be efficiently converted by translation methods via TGBA, such as LTL2BA, Spot and LTL3BA. The number of states of the smallest NBA for the formulae $E_1(n)$, $E_2(n)$ and $C(n)$ are 2^n, $(n+1)^2$ and $n+1$, respectively.

7.2 Discussion

With respect to execution time, our implementation with reduction techniques was much more efficient than LTL2BA and Spot for all of the benchmarks. The execution time of our method was about a tenth that of Spot. For all benchmarks except E_2 and C, our implementation was more efficient than LTL3BA. Furthermore, with respect to the size of the resulting NBA, our implementation with reduction techniques was about as efficient as the other tools. For the benchmarks $E_1(n)$, $E_2(n)$ and $C(n)$, the sizes of the NBAs produced by our implementation were 2^n, $(n+1)^2$ and $n+1$, respectively, which are the smallest sizes that can be expected and are the same sizes that Spot produces. Taken together, these results indicate that with our method one can check the satisfiability of LTL formulae 10 times faster than with Spot, and the NBA obtained by our implementation is suitable for manipulations such as determinisation, complementation, and so forth.

A comparison of our implementations with and without reduction techniques confirmed that the reduction techniques effectively reduced conversion times. From the results of $E_1(n)$, $E_2(n)$ and $U(n)$, it is apparent that the number of states of NBA in our implementation is almost the same with and without the reduction techniques. Furthermore, the conversion times for our implementations with and without reduction techniques were approximately equal. These results indicate that the overhead of our reduction techniques is negligible.

[4] In this benchmark, we did not use the technique introduced in Sect.5.1, since automata from $C(n)$ have too many transitions to be decomposed into SCCs effectively.

Table 3. The number of states and conversion times (s) for E_1, E_2, C and U

	Our implementation				Spot		LTL2BA		LTL3BA	
	with reduction		without reduction		# of		# of		# of	
n	# of states	time	# of states	time	states	time	states	time	states	time
$E_1(5)$	32	0.00	33	0.00	32	0.02	32	0.00	32	0.00
$E_1(8)$	256	0.01	257	0.01	256	0.15	256	0.03	256	0.02
$E_1(11)$	2048	0.13	2049	0.13	2048	6.75	2048	6.34	2048	1.47
$E_1(14)$	16384	3.49	16385	3.57	na	na	na	na	16384	131.86
$E_1(17)$	131072	113.78	131073	106.47	na	na	na	na	na	na
$E_2(11)$	144	0.08	145	0.08	144	0.36	345	0.17	144	0.05
$E_2(18)$	361	1.19	362	1.24	361	4.33	940	3.53	361	0.47
$E_2(25)$	676	9.00	677	9.34	676	27.75	1829	40.66	676	2.86
$E_2(32)$	1089	43.30	1090	45.15	1089	116.46	3012	223.90	1089	10.69
$E_2(39)$	1600	299.31	na	na	na	na	na	na	1600	30.72
$U(4)$	8	0.01	9	0.01	8	0.06	15	0.00	13	0.01
$U(6)$	32	0.01	33	0.01	32	0.05	89	0.01	87	0.02
$U(8)$	128	0.03	129	0.02	128	0.13	481	0.25	479	0.22
$U(10)$	512	0.20	513	0.21	512	1.27	2433	54.27	2431	49.35
$U(12)$	2048	3.33	2049	3.33	2048	23.61	na	na	na	na
$U(14)$	8192	60.41	8193	59.37	na	na	na	na	na	na
$C(3)$	4	0.00	17	0.00	4	0.02	4	0.00	4	0.00
$C(7)$	8	0.00	513	0.04	8	0.17	8	0.27	8	0.01
$C(11)$	12	0.03	12289	17.58	12	0.07	na	na	12	0.02
$C(15)$	16	0.47	na	na	16	0.89	na	na	16	0.17
$C(19)$	20	11.38	na	na	20	29.19	na	na	20	4.78
$C(23)$	na	na	na	na	na	na	na	na	24	180.70

Finally, we discuss the reason why our implementation is more efficient than the implementation of Spot [5], even though our implementation and Spot both utilise BDD for representing transitions. Spot converts LTL formulae into NBA via TGBA, and indicates by BDD whether each transition is included in the acceptance conditions (a set of sets of final transitions). In contrast, our method converts LTL formulae into NBA via GBA, and it is not necessary to represent the acceptance conditions (a set of sets of final states) by BDD. Because the number of transitions is much larger than the number of states, the size of the BDD tends to be larger in Spot than in our method. This difference is the reason why our implementation is more efficient than Spot. LTL3BA also uses BDD for representing transitions from a state. However, from [4], it is not clear if LTL3BA indicates by BDD whether each transition is included in the acceptance conditions.

8 Conclusion

In this paper, we proposed an efficient method for translating LTL formulae into NBA. We also compared our method to LTL2BA, Spot and LTL3BA, which are currently the most efficient tools available. We determined that our method

executes much faster than LTL2BA and Spot and is competitive with LTL3BA. Furthermore, the size of the resulting NBA generated by our method is comparable to that generated by the others. These results show that our method of translation via GBA is competitive with the most efficient currently available tools.

Our method does not implement several reduction techniques that are implemented in Spot or LTL3BA. Future work will integrate these reduction techniques into our method and will also evaluate the method by applying it to actual verification processes, such as model-checking, realisability checking, and so forth.

Methods for converting specifications written in LTL into NBA are commonly used in verification processes. We believe that our method will be of practical use for the verification of safety critical systems.

Acknowledgment. This work was supported by JSPS KAKENHI Grant Number 24500032. We would like to thank the reviewers for their valuable comments and suggestions to improve the quality of the paper.

References

1. Abadi, M., Lamport, L., Wolper, P.: Realizable and unrealizable specifications of reactive systems. In: Ronchi Della Rocca, S., Ausiello, G., Dezani-Ciancaglini, M. (eds.) ICALP 1989. LNCS, vol. 372, pp. 1–17. Springer, Heidelberg (1989)
2. Aoshima, T., Sakuma, K., Yonezaki, N.: An efficient verification procedure supporting evolution of reactive system specifications. In: Proc. of the 4th International Workshop on Principles of Software Evolution, pp. 182–185. ACM (2001)
3. Aoshima, T., Yonezaki, N.: Verification of reactive system specification with outer event conditional formula. In: International Symposium on Principles of Software Evolution (ISPSE2000), pp. 195–199 (2000)
4. Babiak, T., Křetínský, M., Řehák, V., Strejček, J.: LTL to büchi automata translation: Fast and more deterministic. In: Flanagan, C., König, B. (eds.) TACAS 2012. LNCS, vol. 7214, pp. 95–109. Springer, Heidelberg (2012)
5. Duret-Lutz, A.: LTL translation improvements in Spot. In: Proc. of the Fifth international conference on Verification and Evaluation of Computer and Communication Systems, VECoS 2011, pp. 72–83. British Computer Society (2011)
6. Duret-Lutz, A., Poitrenaud, D.: Spot: An extensible model checking library using transition-based generalized Büchi automata. In: Proc. of MASCOTS 2004, pp. 76–83. IEEE Computer Society (2004)
7. Filiot, E., Jin, N., Raskin, J.F.: An antichain algorithm for LTL realizability. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 263–277. Springer, Heidelberg (2009)
8. Gastin, P., Oddoux, D.: Fast LTL to büchi automata translation. In: Berry, G., Comon, H., Finkel, A. (eds.) CAV 2001. LNCS, vol. 2102, pp. 53–65. Springer, Heidelberg (2001)
9. Gerth, R., Peled, D., Vardi, M.Y., Wolper, P.: Simple on-the-fly automatic verification of linear temporal logic. In: Protocol Specification Testing and Verification, pp. 3–18. Chapman & Hall (1995)

10. Giannakopoulou, D., Lerda, F.: From states to transitions: Improving translation of LTL formulae to Büchi automata. In: Peled, D.A., Vardi, M.Y. (eds.) FORTE 2002. LNCS, vol. 2529, pp. 308–326. Springer, Heidelberg (2002)
11. Holzmann, G.J.: The model checker SPIN. IEEE Trans. Softw. Eng. 23(5), 279–295 (1997), http://dx.doi.org/10.1109/32.588521
12. Jobstmann, B., Bloem, R.: Optimizations for LTL synthesis. In: Formal Methods in Computer Aided Design, FMCAD 2006, pp. 117–124 (2006)
13. Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: POPL 1989, pp. 179–190 (1989)
14. Rozier, K.Y., Vardi, M.Y.: LTL satisfiability checking. In: Bošnački, D., Edelkamp, S. (eds.) SPIN 2007. LNCS, vol. 4595, pp. 149–167. Springer, Heidelberg (2007)
15. Somenzi, F., Bloem, R.: Efficient Büchi automata from LTL formulae. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 248–263. Springer, Heidelberg (2000)