Math. Struct. in Comp. Science (2012), *vol.* 22, *pp.* 203–235. © Cambridge University Press 2012
doi:10.1017/S0960129511000399

Almost linear Büchi automata

TOMÁŠ BABIAK[†], VOJTĚCH ŘEHÁK[‡] and JAN STREJČEK[§]

Faculty of Informatics, Masaryk University, Brno, Czech Republic
Email: {xbabiak;rehak;strejcek}@fi.muni.cz

Received 25 January 2010; revised 13 September 2010

We introduce a new fragment of linear temporal logic (LTL) called *LIO* and a new class of Büchi automata (BA) called *almost linear Büchi automata* (ALBA). We provide effective translations between LIO and ALBA showing that the two formalisms are expressively equivalent. As we expect there to be applications of our results in model checking, we use two standard sources of specification formulae, namely Spec Patterns and BEEM, to study the practical relevance of the LIO fragment, and to compare our translation of LIO to ALBA with two standard translations of LTL to BA using alternating automata. Finally, we demonstrate that the LIO to ALBA translation can be much faster than the standard translation, and the resulting automata can be substantially smaller.

1. Introduction

The growing number of concurrent software and hardware systems puts increasing emphasis on the development of automatic verification methods that can be applied in practice. One of the most promising methods is LTL model checking. The main problem with this verification method is the *state explosion problem*, and the consequent high computational complexity. While symbolic approaches to model checking partly solve the problem for hardware systems, there is still no satisfactory solution for the model checking of software systems. The most promising approach is a combination of abstraction methods, reduction methods and optimised model-checking algorithms.

Reduction methods and optimised algorithms are often based on some specific properties of the specification formula or the model. For example, a very effective reduction method called *partial order reduction* employs the fact that in most cases specification formulae do not use the modality *next* and thus describe *stutter-invariant* properties (Lamport 1983). Another example can be found in Černá and Pelánek (2003), where the authors show that two classes of Manna and Pnueli's hierarchy of temporal properties (Manna and Pnueli 1990), namely *guarantee* and *persistence* formulae, can be translated into *terminal* and *weak* Büchi automata, respectively. Černá and Pelánek (2003) also

[†] Tomáš Babiak has been supported by the Czech Science Foundation, grants No. 201/09/1389 and No. 102/09/H042.
[‡] Vojtěch Řehák has been supported by the research centre 'Institute for Theoretical Computer Science (ITI)', project No. 1M0545, and by the Czech Science Foundation, grants No. 201/08/P459 and No. P202/10/1469.
[§] Jan Strejček has been supported by the Ministry of Education of the Czech Republic, project No. MSM0021622419, and by the Czech Science Foundation, grants No. 201/08/P375 and No. P202/10/1469.

https://doi.org/10.1017/S0960129511000399 Published online by Cambridge University Press

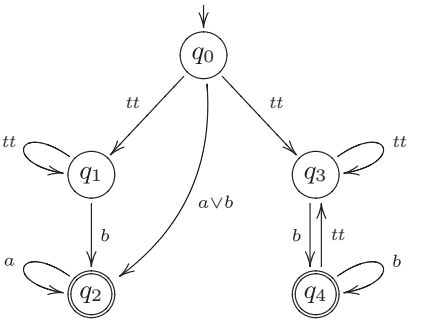


Fig. 1. An ALBA for the formula $\mathsf{G}(a \vee \mathsf{F}b)$. The automaton has two terminal strongly connected components: $\{q_2\}$ corresponding to $\mathsf{G}a$ and $\{q_3, q_4\}$ corresponding to $\mathsf{GF}b$.

suggests several improvements of standard model-checking algorithms employing the specific structure of these automata.

We have also recognised that all formulae of the *restricted temporal logic* (Perrin and Pin 2004), that is, formulae that only use the temporal operators *eventually* (F) and *always* (G), can be translated to Büchi automata (BA) that are linear (1-weak), with the possible exception of terminal strongly connected components. These terminal components also have a specific property: they only accept infinite words over a set of letters, where some selected letters appear infinitely often. We call such automata *almost linear Büchi automata (ALBA)*. Figure 1 provides an example of an ALBA automaton corresponding to the formula $\mathsf{G}(a \vee \mathsf{F}b)$.

We believe that the specific shape of ALBA automata has the potential to improve the model-checking process, especially when terminal strongly connected components are described purely in terms of the sets of letters mentioned above. We can already provide an example of an improvement in sanity checking – sanity checks try to detect basic errors in specifications and system models. For example, a correct specification formula and its negation should both be satisfiable (Rozier and Vardi 2007). In a standard approach to LTL satisfiability checking, the formula is translated into a Büchi automaton and Tarjan's algorithm (Tarjan 1972) or Nested Depth First Search (Nested-DFS) (Courcoubetis *et al.* 1992; Holzmann *et al.* 1996) then decides whether the automaton accepts some word or not. If the formula is translated into an ALBA instead of a general BA, we can use an arbitrary reachability algorithm to decide the satisfiability (basically, we only need to check the reachability of a terminal component since the non-emptiness check for a terminal component is trivial). The asymptotic complexity of Tarjan's algorithm, Nested-DFS and all reachability algorithms is the same, namely, linear. The improvement lies in the fact that some reachability algorithms can be run effectively in parallel and distributed environments, while Tarjan's algorithm and Nested-DFS cannot, since they are based on an intrinsically sequential depth-first search.

Searching for the precise class of LTL formulae corresponding to ALBA has resulted in the definition of an LTL fragment, which we call LIO (the abbreviation for *linear*

https://doi.org/10.1017/S0960129511000399 Published online by Cambridge University Press

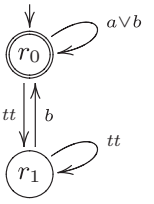


Fig. 2. The Büchi automaton for the formula $\mathsf{G}(a \vee \mathsf{F}b)$ produced by `ltl2ba` (Gastin and Oddoux 2001).

and *infinitely often*). The fragment is strictly more expressive than the restricted temporal logic. To prove that LIO corresponds to ALBA, we present translations between LIO and ALBA.

We will also compare the LIO to ALBA translation with standard translations of LTL formulae to Büchi automata (BA). The main theoretical difference lies in the sizes of the automata produced: while standard translations produce automata with at most exponentially many states (in the length of input formulae), LIO to ALBA can produce double exponential automata. We currently do not know whether this exponential gap is an unavoidable price for the specific form of automata produced or is just a weakness of our translation. However, there are LIO formulae for which the automata created by the standard translations are not ALBA. For example, `ltl2ba` (Gastin and Oddoux 2001) translates the formula $\mathsf{G}(a \vee \mathsf{F}b)$ into the automaton shown in Figure 2, which is not ALBA (if we switch off all optimisations, `ltl2ba` produces an automaton with four states, which is also not ALBA).

In order to get a more realistic view of the relevance and applicability in practice of the LIO to ALBA translation, we have implemented the translation. We have compared this implementation, which we have called `lio2alba`, with two standard LTL to BA translators, namely, `ltl2ba` (Gastin and Oddoux 2001) and the translation used in the distributed model checker DiVinE (Barnat *et al.* 2006). In carrying out the comparison, we used LTL specification formulae taken from two standard sources: Spec Patterns (Dwyer *et al.* 1998) and BEEM (Pelánek 2007). The tests showed that the LIO to ALBA translation can be carried out for the majority of these specification formulae, and the resulting ALBA automata have more or less the same sizes as the automata produced by the reference translators. To compare the efficiency for bigger formulae, we also ran the three translators on some parametrised formulae, and, despite the double exponential theoretical complexity, `lio2alba` turns out to be surprisingly powerful in some cases. For example, the formula

$$\theta_n = \neg((\mathsf{GF}p_1 \wedge \mathsf{GF}p_2 \wedge \ldots \wedge \mathsf{GF}p_n) \rightarrow \mathsf{G}(p \rightarrow \mathsf{F}r))$$

is translated by `lio2alba` for $n = 320$ in approximately the same time needed by `ltl2ba` to translate the formula for $n = 10$. Note that the formula θ_n was taken from the introduction of Gastin and Oddoux (2001), where it was used to demonstrate the efficiency of `ltl2ba`.

https://doi.org/10.1017/S0960129511000399 Published online by Cambridge University Press

Structure of the paper

Section 2 provides the definitions of LTL, LIO, BA and ALBA. Section 3 then describes the ALBA to LIO translation. Section 4 presents the LIO to ALBA translation and the proof of its double exponential complexity. Section 5 describes the `lio2alba` implementation (including some optimisations) and presents the results of the experiments comparing the three implementations. The final section presents conclusions and mentions some topics for future research.

Some of the results in this paper, including a preliminary version of the LIO to ALBA translation with a triple exponential bound, were previously presented in Babiak *et al.* (2009). The detailed results of our experiments can be found in Babiak (2010).

2. Preliminaries

In this section we first recall the definitions of LTL and Büchi automata, and then go on to define the LTL fragment LIO and Almost linear Büchi automata. Finally, we present a hierarchy of language classes corresponding to various types of Büchi automata.

2.1. *Linear temporal logic*

The syntax of *Linear Temporal Logic* (LTL) (Pnueli 1977) is defined by

$$\varphi \ ::= \ tt \ \mid \ a \ \mid \ \neg\varphi \ \mid \ \varphi \vee \varphi \ \mid \ \varphi \wedge \varphi \ \mid \ \mathsf{X}\varphi \ \mid \ \varphi \,\mathsf{U}\, \varphi,$$

where tt stands for *true*, a ranges over a countable set AP of *atomic propositions*, and X and U are modal operators called *next* and *until*, respectively. The logic is interpreted over infinite words over the alphabet $\Sigma = 2^{AP'}$, where $AP' \subseteq AP$ is a finite subset. Given a word $u = u(0)u(1)u(2)\ldots \in (2^{AP'})^\omega$, we use u_i to denote the ith suffix of u, that is, $u_i = u(i)u(i+1)\ldots$.

The semantics of LTL formulae is defined inductively as follows:

$u \models tt$
$u \models a$ iff $a \in u(0)$
$u \models \neg\varphi$ iff $u \not\models \varphi$
$u \models \varphi_1 \vee \varphi_2$ iff $u \models \varphi_1$ or $u \models \varphi_2$
$u \models \varphi_1 \wedge \varphi_2$ iff $u \models \varphi_1$ and $u \models \varphi_2$
$u \models \mathsf{X}\varphi$ iff $u_1 \models \varphi$
$u \models \varphi_1 \,\mathsf{U}\, \varphi_2$ iff $\exists i \geqslant 0 \,.\, (\, u_i \models \varphi_2$ and $\forall\, 0 \leqslant j < i \,.\, u_j \models \varphi_1\,)$.

We say that a word u *satisfies* φ whenever $u \models \varphi$. Given an alphabet Σ, a formula φ defines the language

$$L^\Sigma(\varphi) = \{u \in \Sigma^\omega \mid u \models \varphi\}.$$

We often write $L(\varphi)$ instead of $L^{2^{AP(\varphi)}}(\varphi)$, where $AP(\varphi)$ denotes the set of atomic propositions occurring in the formula φ.

We extend the LTL with derived modal operators:

— $\mathsf{F}\varphi$ called *eventually* and equivalent to $tt \,\mathsf{U}\, \varphi$;

https://doi.org/10.1017/S0960129511000399 Published online by Cambridge University Press

— $\mathsf{G}\varphi$ called *always* and equivalent to $\neg\mathsf{F}\neg\varphi$;
— $\varphi\,\mathsf{R}\,\psi$ called *release* and equivalent to $\neg(\neg\varphi\ \mathsf{U}\ \neg\psi)$; and
— $\varphi\,\mathsf{W}\,\psi$ called *weak until* and equivalent to $(\mathsf{G}\varphi)\vee(\varphi\,\mathsf{U}\,\psi)$.

For a set $\{O_1,\ldots,O_n\}$ of modal operators, LTL$(O_1,\ldots,O_n)$ denotes the LTL fragment containing all formulae with modalities $O_1,\ldots,O_n$ only. We will mainly use the fragments LTL(F,G) with modalities eventually and always, and LTL() without any modalities. In the following, we use $\alpha,\alpha_0,\alpha_1,\ldots$ to represent formulae of LTL(). Note that an LTL() formula describes only a property of the first letter of an infinite word. Hence, we say that a letter $e\in\Sigma$ satisfies an LTL() formula α, written $e\models\alpha$, if and only if $ew\models\alpha$ for some $w\in\Sigma^\omega$.

2.2. *Büchi automata*

Definition 1. A *Büchi automaton* (BA) is a tuple $A=(Q,\Sigma,\delta,q_0,F)$, where:

— Q is a finite set of *states*;
— Σ is a finite *alphabet*;
— $\delta: Q\times\Sigma\to 2^Q$ is a total *transition function*;
— $q_0\in Q$ is an *initial state*; and
— $F\subseteq Q$ is a set of *accepting states.*

We will write $p\xrightarrow{e}q$ instead of $q\in\delta(p,e)$. A Büchi automaton is traditionally seen as a directed graph where nodes are the states and there is an edge leading from p to q and labelled by e whenever $p\xrightarrow{e}q$. An edge $p\xrightarrow{e}p$ is called a *loop* on p.

A *run* π over an infinite word $u(0)u(1)u(2)\ldots\in\Sigma^\omega$ is a sequence

$$\pi = r_0\xrightarrow{u(0)}r_1\xrightarrow{u(1)}r_2\xrightarrow{u(2)}\ldots$$

where $r_0=q_0$ is the initial state. The run is *accepting* if some accepting state occurs infinitely often in the sequence $r_0,r_1,\ldots$. The *language* $L(A)$ defined by automaton A is the set of all infinite words u such that the automaton has an accepting run over u.

A state q is *reachable* from p, written $p\to^* q$, if $p=q$ or there exists a sequence

$$r_0\xrightarrow{u(0)}r_1\xrightarrow{u(1)}r_2\xrightarrow{u(2)}\ldots\xrightarrow{u(n)}r_{n+1}$$

where $p=r_0$ and $q=r_{n+1}$.

A *strongly connected component* (SCC or *component* for short) is a maximal set of states $S\subseteq Q$ such that $p\to^* q$ holds for every $p,q\in S$. Note that every state of an automaton belongs to exactly one strongly connected component.

Several special classes of Büchi automata have already been considered in the context of model checking. A Büchi automaton (Q,Σ,δ,q_0,F) is said to be:

— *terminal* if for each $p\in F$ and $a\in\Sigma$ we have $\delta(p,a)\neq\varnothing$ and $\delta(p,a)\subseteq F$;
— *weak* if every SCC of the automaton either contains only accepting states or only non-accepting states;
— *k-weak* for some $k>0$ if it is weak and every SCC contains at most k states;
— *linear* or *very weak* if it is 1-weak.

https://doi.org/10.1017/S0960129511000399 Published online by Cambridge University Press

Linear Büchi automata can also be defined as automata where each SCC consists of one state, that is, each cycle is a loop.

Given an automaton A and its state q, we use A_q to denote the automaton A where the initial state is changed to q. A strongly connected component S is said to be *terminal* if for all $p \in S$, we have $p \rightarrow^* q$ implies $q \in S$.

In the following, we assume that Büchi automata use alphabets of the form $2^{AP'}$ for some finite set of atomic propositions $AP' \subseteq AP$. When we draw such an automaton, we usually label transitions with LTL() formulae, where $p \xrightarrow{\alpha} q$ means that there is a transition $p \xrightarrow{e} q$ for each $e \in 2^{AP'}$ satisfying the formula α.

2.3. *The LIO fragment*

The *LIO* fragment of LTL is defined at the syntactic level by

$$\varphi \ ::= \ \psi \ \mid \ \varphi \vee \varphi \ \mid \ \varphi \wedge \varphi \ \mid \ \mathsf{X}\varphi \ \mid \ \alpha \,\mathsf{U}\, \varphi,$$

where α ranges over LTL() and ψ ranges over LTL(F, G), that is, ψ is defined by

$$\psi \ ::= \ \alpha \ \mid \ \neg\psi \ \mid \ \psi \vee \psi \ \mid \ \psi \wedge \psi \ \mid \ \mathsf{F}\psi \ \mid \ \mathsf{G}\psi.$$

This fragment does not fit into any standard taxonomy of LTL fragments (Strejček 2004), but it is a strictly more expressive generalisation of the fragment LTL(F, G), which is also known as *restricted temporal logic* (Perrin and Pin 2004). Note that LTL(F, G) covers many of the specification formulae that are frequently used in the context of model checking, for example, typical response formulae of the form $\mathsf{G}(a \Rightarrow \mathsf{F}b)$. In fact, it is more important that LIO contains negations of these formulae, as only the negations of specification formulae need to be translated into automata in model-checking algorithms.

The syntax of LIO can be also extended with other operators that do not modify its expressive power. For example, we can safely add formulae of the form $\varphi \,\mathsf{R}\, \alpha$ and $\alpha \,\mathsf{W}\, \varphi$ since $\varphi \,\mathsf{R}\, \alpha \equiv \mathsf{G}\alpha \vee \alpha \,\mathsf{U}\, (\alpha \wedge \varphi)$ and $\alpha \,\mathsf{W}\, \varphi \equiv \mathsf{G}\alpha \vee \alpha \,\mathsf{U}\, \varphi$.

2.4. *Almost linear Büchi automata*

Definition 2. *An almost linear Büchi automaton* (ALBA) is a Büchi automaton A over an alphabet $\Sigma = 2^{AP'}$ such that every non-terminal SCC contains just one state and for every terminal component S there exists a formula

$$\rho = \mathsf{G}\alpha_0 \ \wedge \bigwedge_{1 \leqslant i \leqslant n} \mathsf{GF}\alpha_i$$

such that $n \geqslant 0$, $\alpha_0, \alpha_1, \ldots, \alpha_n \in$ LTL(), and for every $q \in S$, we have $L(A_q) = L^{\Sigma}(\rho)$, that is, each state of the component S accepts words satisfying ρ exactly.

Note that our condition on terminal components is formulated in purely semantic terms: it does not describe the concrete structure of terminal components. In fact, a formula $\mathsf{G}\alpha_0 \wedge \bigwedge_{0<i\leqslant n} \mathsf{GF}\alpha_i$ can be translated into a (Büchi automaton with a single) component in at least three reasonable ways, which we can illustrate by automata corresponding to the formula $\rho = \mathsf{G}tt \ \wedge \ \mathsf{GF}a_1 \ \wedge \ \mathsf{GF}a_2$:

https://doi.org/10.1017/S0960129511000399 Published online by Cambridge University Press

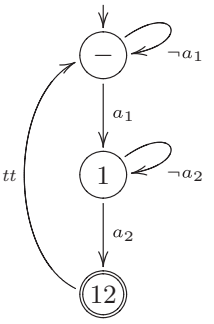


Fig. 3. Minimal number of states and transitions.

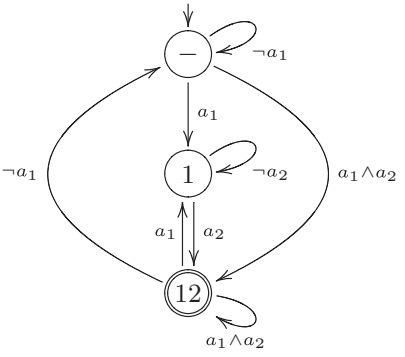


Fig. 4. Minimal number of states and shortcuts.

(1) If we want to minimise the number of transitions and states of the automaton, we just create a 'cycle', as shown in Figure 3.

(2) In the context of LTL model checking, a Büchi automaton A derived from an LTL formula is usually used to build a *product automaton* that accepts all words accepted by A and corresponding to some behaviour of the verified system. Model-checking algorithms then decide whether there is an accepting cycle in the product automaton or not. If we want to keep the number of states of A minimal and shorten the length of potential cycles in product automata, we add some shortcuts to the automaton A, as in Figure 4.

(3) If we want to minimise the length of potential cycles in product automata without regard to the number of states, we translate the formula ρ into the automaton given in Figure 5. Note that the number of states is exponential in the length of ρ, while it is only linear in the previous two cases.

Any of these three shapes of terminal component can be used to formulate an alternative, purely syntactic definition of ALBA. Compared with the original definition, such a syntactic definition would generate a strictly smaller, but expressively equivalent, class of automata.

https://doi.org/10.1017/S0960129511000399 Published online by Cambridge University Press

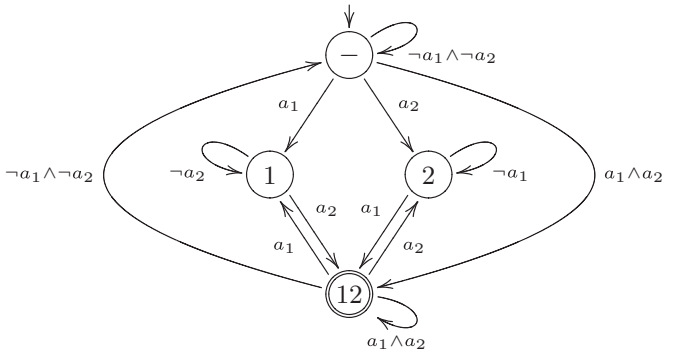


Fig. 5. Shortest cycles in product automata.

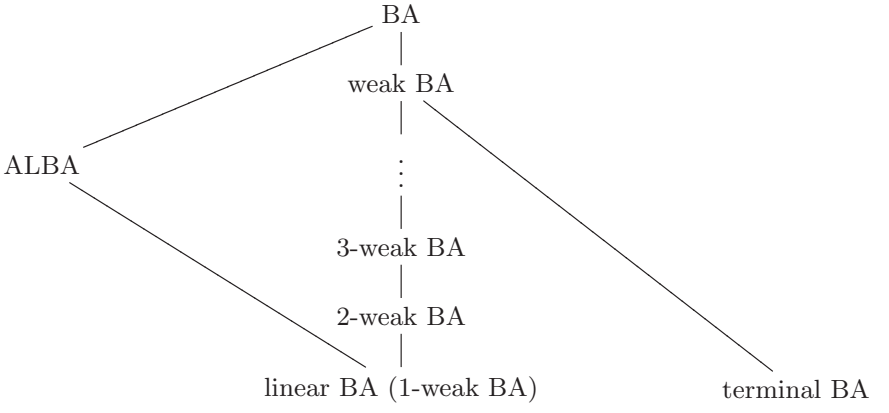


Fig. 6. Hierarchy of Büchi automata classes.

2.5. *Hierarchy of Büchi automata classes*

Figure 6 shows the hierarchy of the classes of Büchi automata described above. A line between two classes means that the upper class is strictly more expressive than the lower class. If the figure does not indicate such a relation between a pair of classes, then the two classes are incomparable.

The inclusions shown follow directly from the definitions of the classes. The strictness of these inclusions is easy to prove, as are the incomparability relations. For example, incomparability of (k-)weak BA (for $k \geqslant 2$) and ALBA classes is due to the following two observations:

(1) One can easily see that only two of the considered automata classes can express the language defined by the formula $\mathsf{GF}a$, namely, ALBA and the general class. Hence, the ALBA class is not included in any of the other classes apart from the general one.
(2) The 2-weak BA of Figure 7 is not equivalent to any ALBA automaton. This follows from the fact that the automaton accepts some words with the suffix $(\{a\}.\varnothing.\{b\})^\omega$, while it does not accept any word with the suffix $(\{a\}.\{b\}.\varnothing)^\omega$. Such a language cannot

https://doi.org/10.1017/S0960129511000399 Published online by Cambridge University Press

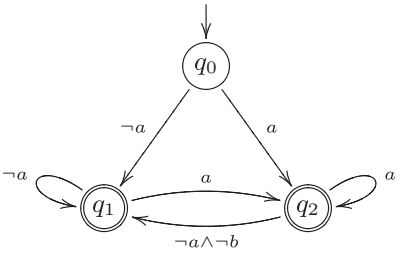


Fig. 7. A 2-weak BA corresponding to the formula $\neg \mathsf{F}(a \wedge (a \,\mathsf{U}\, (b \wedge \neg a)))$.

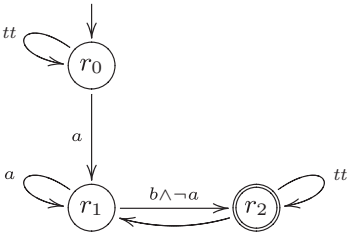


Fig. 8. An ALBA corresponding to the formula $\mathsf{F}(a \wedge (a \,\mathsf{U}\, (b \wedge \neg a)))$.

be recognised by any ALBA since no terminal strongly connected component of an ALBA can distinguish between words that differ only in the order of letters. Hence, the ALBA class does not include any of the (k-)weak automata (for $k \geqslant 2$) classes.

Note that the complement of the language accepted by the 2-weak BA of Figure 7 is accepted by the ALBA automaton given in Figure 8. Hence, the class of ALBA automata is not closed under complementation.

There is also a relation between the ALBA class and the fragment $\mathrm{LTL}^{\mathrm{det}}$, which is better known as *the common fragment of CTL and LTL* (Maidl 2000). As negations of $\mathrm{LTL}^{\mathrm{det}}$ formulae are expressively equivalent to linear Büchi automata (Maidl 2000) and ALBA is an extension of linear BA, we get that ALBA automata are strictly more expressive than negated $\mathrm{LTL}^{\mathrm{det}}$ formulae. Since we will show that ALBA and LIO are equivalent, we can derive the fact that LIO is also strictly more expressive than negated $\mathrm{LTL}^{\mathrm{det}}$ formulae.

3. The ALBA to LIO translation

The translation of an ALBA to LIO formulae is straightforward. Let $A = (Q, \Sigma, \delta, q_0, F)$ be an ALBA over an alphabet $\Sigma = 2^{AP'}$. For every state $q \in Q$, we recursively define an LIO formula $\varphi(q)$ such that $L(A_q) = L^{\Sigma}(\varphi(q))$. There are two cases:

(1) *q is in a terminal strongly connected component*:

https://doi.org/10.1017/S0960129511000399 Published online by Cambridge University Press

From the definition of ALBA, there exists a formula

$$\rho = \mathsf{G}\alpha_0 \wedge \bigwedge_{1 \leqslant i \leqslant n} \mathsf{GF}\alpha_i$$

such that $n \geqslant 0$, $\alpha_0, \alpha_1, \ldots, \alpha_n \in \mathrm{LTL}()$. We set $\varphi(q) = \rho$. Note that ρ is a formula of $\mathrm{LTL}(\mathsf{F}, \mathsf{G})$.

(2) *q is not in any terminal component*:

Let $q \xrightarrow{a_1} q$, $q \xrightarrow{a_2} q$, …, $q \xrightarrow{a_n} q$ be all of the loops on q and $q \xrightarrow{b_1} q_1$, $q \xrightarrow{b_2} q_2$, …, $q \xrightarrow{b_m} q_m$ be all of the transitions leading from q to other states. For every $a \in \Sigma$, let $\alpha(a)$ be an LTL() formula that is satisfied only by the letter a. Then we set

$$(q) = \begin{cases} \left(\bigvee_{1 \leqslant i \leqslant n} \alpha(a_i) \right) \mathsf{U} \bigvee_{1 \leqslant j \leqslant m} (\alpha(b_j) \wedge \mathsf{X}\varphi(q_j)) & \text{if } q \notin F, \\ \left(\left(\bigvee_{1 \leqslant i \leqslant n} \alpha(a_i) \right) \mathsf{U} \bigvee_{1 \leqslant j \leqslant m} (\alpha(b_j) \wedge \mathsf{X}\varphi(q_j)) \right) \vee \mathsf{G} \bigvee_{0 < i \leqslant n} \alpha(a_i) & \text{if } q \in F. \end{cases}$$

Note that $\varphi(q)$ is an LIO formula if we assume that all $\varphi(q_j)$ are in LIO.

The recursion in the definition of $\varphi(q)$ is bounded as A is linear (except the terminal components). The whole automaton then corresponds to the formula $\varphi(q_0)$. Hence, we can state the following theorem.

Theorem 3. Given an ALBA A over an alphabet $\Sigma = 2^{AP'}$, there exists an LIO formula φ such that

$$L(A) = L^{\Sigma}(\varphi).$$

4. The LIO to ALBA translation

The translation proceeds in two steps:

(1) A given LIO formula is transformed to an equivalent LIO formula in *normal form.*

(2) The formula in normal form is translated into an equivalent ALBA.

These steps are described in Sections 4.1 and 4.2, respectively, and we then analyse the complexity of our translation in Section 4.3.

For each LIO formula φ, we define its *size* as follows.
If φ is in LTL(), we define:

$$size(\varphi) = 1$$

https://doi.org/10.1017/S0960129511000399 Published online by Cambridge University Press

and, otherwise, we define $size(\varphi)$ recursively as follows:

$$\begin{aligned} size(\varphi_1 \vee \varphi_2) &= size(\varphi_1) + 1 + size(\varphi_2) \\ size(\varphi_1 \wedge \varphi_2) &= size(\varphi_1) + 1 + size(\varphi_2) \\ size(\neg\varphi_0) &= 1 + size(\varphi_0) \\ size(\mathsf{F}\varphi_0) &= 1 + size(\varphi_0) \\ size(\mathsf{G}\varphi_0) &= 1 + size(\varphi_0) \\ size(\mathsf{X}\varphi_0) &= 1 + size(\varphi_0) \\ size(\alpha\,\mathsf{U}\,\varphi_0) &= 2 + size(\varphi_0). \end{aligned}$$

In this section, we will always assume that LIO formulae are in *positive form*, that is, no temporal operator lies within the scope of any negation. Every LIO formula φ can be transformed into an equivalent LIO formula φ' in positive form using the following equivalences.

$$\begin{aligned} \neg\mathsf{F}\varphi_0 &\equiv \mathsf{G}\neg\varphi_0 \\ \neg\mathsf{G}\varphi_0 &\equiv \mathsf{F}\neg\varphi_0 \\ \neg(\varphi_1 \wedge \varphi_2) &\equiv \neg\varphi_1 \vee \neg\varphi_2 \\ \neg(\varphi_1 \vee \varphi_2) &\equiv \neg\varphi_1 \wedge \neg\varphi_2. \end{aligned}$$

Note that $size(\varphi') \leqslant size(\varphi)$, that is, the transformation to positive form does not increase the size of LIO formulae.

4.1. *Transformation of LIO formulae to normal form*

We say that an LIO formula φ is in *normal form* if it has one of the following forms:

$$\varphi \;::=\; \psi \;\mid\; \varphi \vee \varphi \;\mid\; \varphi \wedge \varphi \;\mid\; \mathsf{X}\varphi \;\mid\; \alpha\,\mathsf{U}\,\varphi,$$

where α ranges over LTL() and ψ is defined by

$$\psi \;::=\; \alpha \;\mid\; \mathsf{G}\alpha \;\mid\; \mathsf{GF}\alpha \;\mid\; \psi \vee \psi \;\mid\; \psi \wedge \psi \;\mid\; \mathsf{F}\psi.$$

In other words, the normal form says that a formula is in positive form and the modality G can occur only in subformulae of the form $\mathsf{G}\alpha$ or $\mathsf{GF}\alpha$. The LIO formulae in normal form are called *nLIO* formulae.

Note that the definition of normal form only restricts subformulae ψ. Hence, to transform an LIO formula to normal form, it is sufficient to transform its LTL(F, G) subformulae to LIO formulae in normal form. We will assume that LIO formulae are already in positive form. Intuitively, we still need to push the operators G towards the subformulae of the form α or $\mathsf{F}\alpha$. This can be done by repeated application of the following

https://doi.org/10.1017/S0960129511000399 Published online by Cambridge University Press

equivalences:

$$\mathsf{G}(\varphi_1 \wedge \varphi_2) \equiv \mathsf{G}\varphi_1 \wedge \mathsf{G}\varphi_2$$
$$\mathsf{G}(\varphi_1 \vee (\varphi_2 \wedge \varphi_3)) \equiv \mathsf{G}(\varphi_1 \vee \varphi_2) \wedge \mathsf{G}(\varphi_1 \vee \varphi_3)$$
$$\mathsf{G}(\varphi_1 \vee \mathsf{F}\varphi_2) \equiv \mathsf{G}\varphi_1 \vee \mathsf{F}(\varphi_2 \wedge \mathsf{XG}\varphi_1) \vee \mathsf{GF}\varphi_2$$
$$\mathsf{GF}(\varphi_1 \vee \varphi_2) \equiv \mathsf{GF}\varphi_1 \vee \mathsf{GF}\varphi_2$$
$$\mathsf{GF}(\varphi_1 \wedge (\varphi_2 \vee \varphi_3)) \equiv \mathsf{GF}(\varphi_1 \wedge \varphi_2) \vee \mathsf{GF}(\varphi_1 \wedge \varphi_3)$$
$$\mathsf{GF}(\varphi_1 \wedge \mathsf{F}\varphi_2) \equiv \mathsf{GF}\varphi_1 \wedge \mathsf{GF}\varphi_2$$
$$\mathsf{GF}(\varphi_1 \wedge \mathsf{G}\varphi_2) \equiv \mathsf{GF}\varphi_1 \wedge \mathsf{FG}\varphi_2$$
$$\mathsf{GFF}\varphi \equiv \mathsf{GF}\varphi$$
$$\mathsf{GFG}\varphi \equiv \mathsf{FG}\varphi$$
$$\mathsf{GG}\varphi \equiv \mathsf{G}\varphi$$
$$\mathsf{G}\left(\bigvee_{\varphi \in G} \mathsf{G}\varphi\right) \equiv \bigvee_{\varphi \in G} \mathsf{G}\varphi$$
$$\mathsf{G}\left(\alpha \vee \bigvee_{\varphi \in G} \mathsf{G}\varphi\right) \equiv \mathsf{G}\alpha \vee \alpha\,\mathsf{U}\left(\bigvee_{\varphi \in G} \mathsf{G}\varphi\right)$$

Lemma 4. For every formula φ of LTL(F, G), we can effectively construct an equivalent nLIO formula.

Proof. For a given LTL(F, G) formula φ in positive form, we construct an equivalent LIO formula $\mathbf{nf}(\varphi)$ in normal form. The formula $\mathbf{nf}(\varphi)$ is defined recursively. The recursion is bounded as each $\mathbf{nf}(\varphi')$ appearing in the definition of $\mathbf{nf}(\varphi)$ satisfies $size(\varphi') < size(\varphi)$. We define $\mathbf{nf}(\varphi)$ according to the structure of φ.

— α:

$$\mathbf{nf}(\alpha) = \alpha.$$

In the remaining cases we assume that $\varphi \notin \text{LTL}()$.

— $\varphi_1 \vee \varphi_2$:

$$\mathbf{nf}(\varphi_1 \vee \varphi_2) = \mathbf{nf}(\varphi_1) \vee \mathbf{nf}(\varphi_2).$$

— $\varphi_1 \wedge \varphi_2$:

$$\mathbf{nf}(\varphi_1 \wedge \varphi_2) = \mathbf{nf}(\varphi_1) \wedge \mathbf{nf}(\varphi_2).$$

— $\mathsf{F}\varphi_0$:

$$\mathbf{nf}(\mathsf{F}\varphi_0) = tt\,\mathsf{U}\,(\mathbf{nf}(\varphi_0)).$$

— $\mathsf{G}\varphi_0$:

This case is divided into the following subcases according to the structure of φ_0:

– α: $\mathbf{nf}(\mathsf{G}\alpha) = \mathsf{G}\alpha$

In the remaining cases we assume that $\varphi_0 \notin \text{LTL}()$.

https://doi.org/10.1017/S0960129511000399 Published online by Cambridge University Press

– $\varphi_1 \wedge \varphi_2$:

$$\mathbf{nf}(\mathsf{G}(\varphi_1 \wedge \varphi_2)) = \mathbf{nf}(\mathsf{G}\varphi_1) \wedge \mathbf{nf}(\mathsf{G}\varphi_2)$$

– $\mathsf{F}\varphi_1$:

This case is again divided into the following subcases according to the structure of φ_1:

- α:

$$\mathbf{nf}(\mathsf{GF}\alpha) = \mathsf{GF}\alpha$$

In the remaining cases we assume that $\varphi_1 \notin \mathrm{LTL}()$.

- $\varphi_3 \vee \varphi_4$:

$$\mathbf{nf}(\mathsf{GF}(\varphi_3 \vee \varphi_4)) = \mathbf{nf}(\mathsf{GF}\varphi_3) \vee \mathbf{nf}(\mathsf{GF}\varphi_4).$$

- $\varphi_3 \wedge \varphi_4$:

As conjunction is an associative operator, we can treat it as an operator of arbitrary arity and assume that none of the conjuncts are conjunctions. Then either all conjuncts are formulae of LTL() (that is, $\varphi_3 \wedge \varphi_4 \in \mathrm{LTL}()$, but this case was covered by the $\mathsf{GF}\alpha$ case), or at least one of the conjuncts has the form $\varphi_5 \vee \varphi_6$ or $\mathsf{F}\varphi_5$ or $\mathsf{G}\varphi_5$. Let φ_4 be this conjunct and φ_3 be the conjunction of all the other conjuncts. We proceed according to the structure of φ_4:

- $\varphi_5 \vee \varphi_6$:

Since

$$\mathsf{GF}(\varphi_3 \wedge (\varphi_5 \vee \varphi_6)) \equiv \mathsf{GF}(\varphi_3 \wedge \varphi_5) \vee \mathsf{GF}(\varphi_3 \wedge \varphi_6),$$

we set

$$\mathbf{nf}(\mathsf{GF}(\varphi_3 \wedge (\varphi_5 \vee \varphi_6))) = \mathbf{nf}(\mathsf{GF}(\varphi_3 \wedge \varphi_5)) \vee \mathbf{nf}(\mathsf{GF}(\varphi_3 \wedge \varphi_6)).$$

- $\mathsf{F}\varphi_5$:

Since

$$\mathsf{GF}(\varphi_3 \wedge \mathsf{F}\varphi_5) \equiv (\mathsf{GF}\varphi_3) \wedge (\mathsf{GF}\varphi_5),$$

we set

$$\mathbf{nf}(\mathsf{GF}(\varphi_3 \wedge \mathsf{F}\varphi_5)) = \mathbf{nf}(\mathsf{GF}\varphi_3) \wedge \mathbf{nf}(\mathsf{GF}\varphi_5).$$

- $\mathsf{G}\varphi_5$:

Since

$$\mathsf{GF}(\varphi_3 \wedge \mathsf{G}\varphi_5) \equiv (\mathsf{GF}\varphi_3) \wedge (\mathsf{FG}\varphi_5),$$

we set

$$\mathbf{nf}(\mathsf{GF}(\varphi_3 \wedge \mathsf{G}\varphi_5)) = \mathbf{nf}(\mathsf{GF}\varphi_3) \wedge tt\,\mathsf{U}\,(\mathbf{nf}(\mathsf{G}\varphi_5)).$$

- $\mathsf{F}\varphi_3$:

$$\mathbf{nf}(\mathsf{GFF}\varphi_3) = \mathbf{nf}(\mathsf{GF}\varphi_3).$$

https://doi.org/10.1017/S0960129511000399 Published online by Cambridge University Press

- $\mathsf{G}\varphi_3$:
  Since
  $$\mathsf{GFG}\varphi_3 \equiv \mathsf{FG}\varphi_3,$$
  we set
  $$\mathbf{nf}(\mathsf{GFG}\varphi_3) = tt\,\mathsf{U}\,(\mathbf{nf}(\mathsf{G}\varphi_3)).$$

– $\varphi_1 \vee \varphi_2$:
The situation is similar to the $\mathsf{GF}(\varphi_3 \wedge \varphi_4)$ case, so either $\varphi_1 \vee \varphi_2 \in$ LTL() (which was covered by the $\mathsf{G}\alpha$ case), or we can assume that φ_2 has the form $\varphi_3 \wedge \varphi_4$ or $\mathsf{F}\varphi_3$ or $\mathsf{G}\varphi_3$. We proceed according to the structure of φ_2:

- $\varphi_3 \wedge \varphi_4$:
  Since
  $$\mathsf{G}(\varphi_1 \vee (\varphi_3 \wedge \varphi_4)) \equiv \mathsf{G}(\varphi_1 \vee \varphi_3) \wedge \mathsf{G}(\varphi_1 \vee \varphi_4),$$
  we set
  $$\mathbf{nf}(\mathsf{G}(\varphi_1 \vee (\varphi_3 \wedge \varphi_4))) = \mathbf{nf}(\mathsf{G}(\varphi_1 \vee \varphi_3)) \wedge \mathbf{nf}(\mathsf{G}(\varphi_1 \vee \varphi_4)).$$

- $\mathsf{F}\varphi_3$:
  Since
  $$\mathsf{G}(\varphi_1 \vee \mathsf{F}\varphi_3) \equiv (\mathsf{G}\varphi_1) \vee \mathsf{F}(\varphi_3 \wedge \mathsf{XG}\varphi_1) \vee \mathsf{GF}\varphi_3,$$
  we set
  $$\mathbf{nf}(\mathsf{G}(\varphi_1 \vee \mathsf{F}\varphi_3)) = \mathbf{nf}(\mathsf{G}\varphi_1) \vee tt\,\mathsf{U}\,(\mathbf{nf}(\varphi_3) \wedge \mathsf{X}(\mathbf{nf}(\mathsf{G}\varphi_1))) \vee \mathbf{nf}(\mathsf{GF}\varphi_3).$$

- $\mathsf{G}\varphi_3$:
  $$\mathbf{nf}(\mathsf{G}(\varphi_1 \vee \mathsf{G}\varphi_3)).$$
  We can assume that φ_1 is an LTL() formula or a formula of the form $\mathsf{G}\varphi'$, or a disjunction of such formulae (all other possibilities are covered by the previous two cases). Hence, the whole subformula $\varphi_1 \vee \mathsf{G}\varphi_3$ can be treated as either $\bigvee_{\varphi' \in G} \mathsf{G}\varphi'$ or $\alpha \vee \bigvee_{\varphi' \in G} \mathsf{G}\varphi'$.
  - $\bigvee_{\varphi' \in G} \mathsf{G}\varphi'$:
    Since
    $$\mathsf{G}\left(\bigvee_{\varphi' \in G} \mathsf{G}\varphi'\right) \equiv \bigvee_{\varphi' \in G} (\mathsf{G}\varphi'),$$
    we set
    $$\mathbf{nf}\left(\mathsf{G}\left(\bigvee_{\varphi' \in G} \mathsf{G}\varphi'\right)\right) = \bigvee_{\varphi' \in G} \mathbf{nf}(\mathsf{G}\varphi').$$

https://doi.org/10.1017/S0960129511000399 Published online by Cambridge University Press

- $\alpha \vee \bigvee_{\varphi' \in G} \mathsf{G}\varphi'$:
  Since
  $$\mathsf{G}\left(\alpha \vee \bigvee_{\varphi' \in G} \mathsf{G}\varphi'\right) \equiv (\mathsf{G}\alpha) \vee \left(\alpha \,\mathsf{U}\, \left(\bigvee_{\varphi' \in G} \mathsf{G}\varphi'\right)\right),$$
  we set
  $$\mathbf{nf}\left(\mathsf{G}\left(\alpha \vee \bigvee_{\varphi' \in G} \mathsf{G}\varphi'\right)\right) = (\mathsf{G}\alpha) \vee \left(\alpha \,\mathsf{U}\, \left(\bigvee_{\varphi' \in G} \mathbf{nf}(\mathsf{G}\varphi')\right)\right).$$

– $\mathsf{G}\varphi_1$:
$$\mathbf{nf}(\mathsf{G}\mathsf{G}\varphi_1) = \mathbf{nf}(\mathsf{G}\varphi_1).$$ □

4.2. *Automata construction for LIO formulae in normal form*

States of the constructed automata correspond to finite sets of nLIO formulae. Given an nLIO formula φ_0, the initial state of the corresponding ALBA is the singleton $\{\varphi_0\}$. Transitions and other states of the automaton are computed by a function R. To every nLIO formula φ, the function assigns a finite set $R(\varphi)$ of pairs of the form (α, S), where $\alpha \in \mathrm{LTL}()$ and S is a finite set of nLIO formulae. The set $R(\varphi)$ satisfies
$$\varphi \equiv \bigvee_{(\alpha,S)\in R(\varphi)} \left(\alpha \wedge \mathsf{X} \bigwedge_{\sigma \in S} \sigma\right).$$
In other words, a word u satisfies φ if and only if there is some pair $(\alpha, S) \in R(\varphi)$ such that the first letter of u satisfies α and the suffix u_1 satisfies all nLIO formulae in S. Intuitively, every pair $(\alpha, S) \in R(\varphi)$ encodes a transition $\{\varphi\} \xrightarrow{\alpha} S$, and the set S corresponds semantically to the conjunction of its elements.

The set $R(\varphi)$ is defined recursively according to the structure of φ:
$$R(\alpha) = \{(\alpha, \varnothing)\}$$
in the remaining cases we assume that $\varphi \notin \mathrm{LTL}()$
$$\begin{aligned}
R(\varphi_1 \vee \varphi_2) &= R(\varphi_1) \cup R(\varphi_2)\\
R(\varphi_1 \wedge \varphi_2) &= \{(\alpha_1 \wedge \alpha_2, S_1 \cup S_2) \mid (\alpha_1, S_1) \in R(\varphi_1), (\alpha_2, S_2) \in R(\varphi_2)\}\\
R(\mathsf{F}\varphi_0) &= \{(tt, \{\mathsf{F}\varphi_0\})\} \cup R(\varphi_0)\\
R(\mathsf{X}\varphi_0) &= \{(tt, \{\varphi_0\})\}\\
R(\alpha \,\mathsf{U}\, \varphi_0) &= \{(\alpha, \{\alpha \,\mathsf{U}\, \varphi_0\})\} \cup R(\varphi_0)\\
R(\mathsf{G}\alpha) &= \{(\alpha, \{\mathsf{G}\alpha\})\}\\
R(\mathsf{GF}\alpha) &= \{(tt, \{\mathsf{GF}\alpha\})\}.
\end{aligned}$$

https://doi.org/10.1017/S0960129511000399 Published online by Cambridge University Press

We extend the functions R and *size* to finite sets of nLIO formulae. For every non-empty finite set $S = \{\varphi_1, \varphi_2, \ldots, \varphi_k\}$ of nLIO formulae, we define

$$R(S) = R\left(\bigwedge_{1 \leqslant i \leqslant k} \varphi_i\right)$$
$$size(S) = \sum_{1 \leqslant i \leqslant k} size(\varphi_i)$$

and for the empty set we define

$$R(\varnothing) = \{(tt, \varnothing)\}$$
$$size(\varnothing) = 0.$$

Remark 5. It is easy to confirm that for each $(\alpha, S) \in R(\varphi)$, the set S contains only subformulae of φ (possibly including the whole formula). Moreover, for a non-empty set $S = \{\varphi_1, \varphi_2, \ldots, \varphi_k\}$ of nLIO formulae, we have

$$R(S) = \{(\bigwedge_{1 \leqslant i \leqslant k} \alpha_i, \bigcup_{1 \leqslant i \leqslant k} S_i) \mid (\alpha_i, S_i) \in R(\varphi_i) \text{ for each } 1 \leqslant i \leqslant k\}.$$

Hence, for each $(\alpha, S') \in R(S)$, the set S' only contains subformulae of $\varphi_1, \varphi_2, \ldots, \varphi_k$ (possibly including the whole formulae).

Before we give a precise description of the automata construction, we will formulate and prove three lemmas that are crucial for proving the finiteness and ALBA structure of the constructed automata.

Lemma 6. Let φ be an nLIO formula. For every $(\alpha, S) \in R(\varphi)$, either $S = \{\varphi\}$ or $size(S) < size(\varphi)$.

Proof. Let φ be an nLIO formula and S be a set such that $(\alpha, S) \in R(\varphi)$ for some α. The proof is carried out by induction on $size(\varphi)$:

— If $size(\varphi) = 1$, then $\varphi \in$ LTL(). As $R(\alpha) = \{(\alpha, \varnothing)\}$, we get that $S = \varnothing$ and the statement clearly holds: $size(\varnothing) = 0 < size(\varphi) = 1$.

— If $size(\varphi) > 1$, we distinguish four cases according to the structure of φ:

  – Either $\mathsf{G}\alpha$ or $\mathsf{GF}\alpha$:

    The definition of $R(\varphi)$ implies that $S = \{\varphi\}$.

  – $\varphi_1 \vee \varphi_2$:

    Then S comes from $R(\varphi_1) \cup R(\varphi_2)$. Let us assume that S comes from $R(\varphi_1)$. As $size(\varphi) > 1$, we know that $\varphi \notin$ LTL(). Hence, $size(\varphi_1) < size(\varphi)$ and we can apply induction hypothesis to get $size(S) \leqslant size(\varphi_1)$. This implies $size(S) < size(\varphi)$. The analogous arguments prove the statement for φ of the form $\mathsf{X}\varphi_0$.

  – Either $\mathsf{F}\varphi_0$ or $\alpha\,\mathsf{U}\,\varphi_0$:

    Then either $S = \{\varphi\}$ or S comes from $R(\varphi_0)$ where $size(\varphi_0) < size(\varphi)$ and the statement follows directly from the induction hypothesis.

https://doi.org/10.1017/S0960129511000399 Published online by Cambridge University Press

– $\varphi_1 \wedge \varphi_2$:

Then $S = S_1 \cup S_2$ where $(\alpha_1, S_1) \in R(\varphi_1)$ and $(\alpha_2, S_2) \in R(\varphi_2)$. As $size(\varphi) > 1$, we know that $\varphi \notin \mathrm{LTL}()$ and hence

$$size(\varphi) > size(\varphi_1) + size(\varphi_2).$$

The induction hypothesis gives us $size(S_1) \leqslant size(\varphi_1)$ and $size(S_2) \leqslant size(\varphi_2)$, but since

$$\begin{aligned} size(S) &= size(S_1) + size(S_2) \\ &\leqslant size(\varphi_1) + size(\varphi_2) \\ &< size(\varphi), \end{aligned}$$

we are done. □

Lemma 7. Let S be a finite set of nLIO formulae. For every $(\alpha, S') \in R(S)$, we have $S' = S$ or $size(S') < size(S)$.

Proof. The lemma clearly holds for $S = \varnothing$, so we assume that $S = \{\varphi_1, \varphi_2, \ldots, \varphi_k\}$ is non-empty. The definition of $R(S)$ implies that each S' is of the form $S' = S_1 \cup S_2 \cup \ldots \cup S_k$ where, for every $1 \leqslant i \leqslant k$, we have $(\alpha_i, S_i) \in R(\varphi_i)$ for some α_i. Lemma 6 says that each S_i satisfies either $S_i = \{\varphi_i\}$ or $size(S_i) < size(\varphi_i)$. If $S_i = \{\varphi_i\}$ holds for all S_i, then $S' = S$. Otherwise, $size(S_i) < size(\varphi_i)$ for some S_i, so

$$\begin{aligned} size(S') &\leqslant \sum_{1 \leqslant i \leqslant k} size(S_i) \\ &< \sum_{1 \leqslant i \leqslant k} size(\varphi_i) \\ &= size(S), \end{aligned}$$

and we are done. □

Lemma 8. Let S be a finite set of nLIO formulae. We have

$$S \subseteq \{\mathsf{G}\alpha, \mathsf{GF}\alpha \mid \alpha \in \mathrm{LTL}()\} \quad \Longleftrightarrow \quad \forall (\alpha', S') \in R(S) \,.\, S' = S.$$

Proof.

($\Longrightarrow$) This direction follows immediately from the definition of $R(\mathsf{G}\alpha)$, $R(\mathsf{GF}\alpha)$, and $R(\varphi_1 \wedge \varphi_2)$.

($\Longleftarrow$) We prove the contraposition and assume that there exists a formula σ in

$$S \setminus \{\mathsf{G}\alpha, \mathsf{GF}\alpha \mid \alpha \in \mathrm{LTL}()\}.$$

With Lemma 6 in mind, it is easy to see that the set $R(\sigma)$ contains a pair (α_1, S_1) such that $size(S_1) < size(\sigma)$. We now let (α_2, S_2) be an arbitrary element of $R(S \setminus \{\sigma\})$. Lemma 7 implies that $size(S_2) \leqslant size(S \setminus \{\sigma\})$. Then $R(S)$ contains a

https://doi.org/10.1017/S0960129511000399 Published online by Cambridge University Press

pair $(\alpha_1 \wedge \alpha_2, S_1 \cup S_2)$ and

$$\begin{aligned} size(S_1 \cup S_2) &\leqslant size(S_1) + size(S_2) \\ &< size(\sigma) + size(S \setminus \{\sigma\}) \\ &= size(S). \end{aligned}$$

Hence, $S_1 \cup S_2 \neq S$. □

We are now ready to present the automata construction. Let φ be an nLIO formula. First we construct a labelled transition system $T_\varphi = (Q, \Sigma, \delta)$, where Q is a set of states, Σ is an alphabet of transition labels and $\delta : Q \times \Sigma \to 2^Q$ is a transition function. This transition system is later slightly modified into an ALBA corresponding to φ. The (terminal) strongly connected components of transition systems are defined precisely in the same way as for Büchi automata.

For a given nLIO formula φ, we define a transition system $T_\varphi = (Q, \Sigma, \delta)$, where:

— Q is the smallest subset of 2^{nLIO} satisfying two conditions:
  (*i*) $\{\varphi\} \in Q$.
  (*ii*) For every $S \in Q$, we have $(\alpha, S') \in R(S)$ implies $S' \in Q$.
— $\Sigma = 2^{AP(\varphi)}$.
— For each $e \in \Sigma$ and $S \in Q$, we set $\delta(S, e) = \{S' \mid (\alpha, S') \in R(S) \text{ and } e \models \alpha\}$.

The following lemma summarises the basic properties of T_φ. The lemma is a direct corollary of the properties of the function R and Lemmas 7 and 8.

Lemma 9. Let φ be an nLIO formula. Then the transition system $T_\varphi = (Q, \Sigma, \delta)$ has the following properties:

— For every $S \in Q$ and every word $u = u(0)u(1)u(2)\ldots \in \Sigma^\omega$, we have

$$u \models \bigwedge_{\sigma \in S} \sigma \quad \Longleftrightarrow \quad u_1 \models \bigwedge_{\sigma \in S'} \sigma \text{ for some } S' \in \delta(S, u(0)).$$

— The set Q is finite.
— Every strongly connected component of T_φ contains just one state.
— Every state $S \in Q$ satisfying $S \subseteq \{\mathsf{G}\alpha, \mathsf{GF}\alpha \mid \alpha \in \mathrm{LTL}()\}$ is a terminal strongly connected component and *vice versa*.

The labelled transition system $T_\varphi = (Q, \Sigma, \delta)$ is modified into a Büchi automaton $A_\varphi = (Q', \Sigma, \delta', \{q_0\}, F)$ as follows. Every state $S \subseteq \{\mathsf{G}\alpha, \mathsf{GF}\alpha \mid \alpha \in \mathrm{LTL}()\}$ (that is, every terminal SCC) is replaced by a strongly connected component corresponding to the formula

$$\left(\mathsf{G} \bigwedge_{\mathsf{G}\alpha \in S} \alpha\right) \quad \wedge \bigwedge_{\mathsf{GF}\alpha \in S} \mathsf{GF}\alpha.$$

The new terminal strongly connected components can be constructed, for example, in the style of Figure 4. The set F of accepting states is the union of sets of accepting states of the new terminal strongly connected components.

https://doi.org/10.1017/S0960129511000399 Published online by Cambridge University Press

Theorem 10. Given an nLIO formula φ, we can effectively construct an ALBA A such that $L(\varphi) = L(A)$.

Proof. Let A be the automaton A_φ constructed above. Lemma 9 and the construction of the automaton from T_φ imply that the resulting automaton A is ALBA. It remains to show that $L(\varphi) = L(A)$.

Recall that A_q stands for the automaton A with the initial state changed to the state q. Also, for each state q, we define its *distance to terminal SCCs*, written $dist(q)$, as the maximal length of an acyclic path leading from q to a terminal SCC. In particular, for each state q of a terminal SCCs, we set $dist(q) = 0$.

We will now prove by induction on $dist(q)$ that every state q represents the correct language, that is, $L(A_q) = L(\bigwedge_{\sigma \in S} \sigma)$, where $S = q$ if q is not in terminal SCC; otherwise $S \subseteq \{\mathsf{G}\alpha, \mathsf{GF}\alpha \mid \alpha \in \mathrm{LTL}()\}$ is the state of T_φ corresponding to the terminal SCC containing q.

If $dist(q) = 0$, then q is a state of a terminal SCC. The correctness follows from the modification of T_φ into A_φ and the fact that the following equivalence holds for each $S \subseteq \{\mathsf{G}\alpha, \mathsf{GF}\alpha \mid \alpha \in \mathrm{LTL}()\}$:

$$\bigwedge_{\sigma \in S} \sigma \quad \Longleftrightarrow \quad \left(\mathsf{G} \bigwedge_{\mathsf{G}\alpha \in S} \alpha\right) \quad \wedge \bigwedge_{\mathsf{GF}\alpha \in S} \mathsf{GF}\alpha.$$

If $dist(q) > 0$, then q is directly a finite set S of nLIO formulae. Our induction hypothesis says that every successor q' of S (such that $q' \neq S$) represents the correct language. If there is no loop on S, the relation $L(A_S) = L(\bigwedge_{\sigma \in S} \sigma)$ follows directly from the first property of Lemma 9 and the induction hypothesis. Otherwise, S has a loop and the definition of $R(S)$ implies that

$$S \subseteq \{\mathsf{F}\varphi_0, \alpha \,\mathsf{U}\, \varphi_0, \mathsf{G}\alpha, \mathsf{GF}\alpha \mid \alpha \in \mathrm{LTL}(), \varphi_0 \in \mathrm{nLIO}\}.$$

Also, the construction of the automaton implies that

$$S \not\subseteq \{\mathsf{G}\alpha, \mathsf{GF}\alpha \mid \alpha \in \mathrm{LTL}()\}.$$

The correctness again follows from the induction hypothesis and Lemma 9. □

4.3. *Complexity of the translation*

In this subsection we show that the number of states of the constructed ALBA is at most double exponential in the size of the input LIO formula φ. First, we prove an exponential upper bound on the size of the nLIO formula $\mathbf{nf}(\varphi)$. Then we prove an exponential upper bound on the size of the resulting ALBA in the size of a given nLIO formula. Finally, we provide a parametrised LIO formula showing that the double exponential upper bound is tight up to a constant factor.

Lemma 11. Given a formula φ of LTL(F, G), we can effectively construct an equivalent nLIO formula $\mathbf{nf}(\varphi)$ such that $size(\mathbf{nf}(\varphi)) \leqslant 2^{size(\varphi)}$.

https://doi.org/10.1017/S0960129511000399 Published online by Cambridge University Press

Proof. The proof is again done by induction on the size of φ and exhibits the same structure as the proof of Lemma 4:

— α:

$$\begin{aligned} size(\mathbf{nf}(\alpha)) &= size(\alpha) \\ &= 1 \\ &< 2 \\ &= 2^{size(\alpha)}. \end{aligned}$$

In the remaining cases we assume that $\varphi \notin \mathrm{LTL}()$.

— $\varphi_1 \vee \varphi_2$:

$$\begin{aligned} size(\mathbf{nf}(\varphi_1 \vee \varphi_2)) &= size(\mathbf{nf}(\varphi_1) \vee \mathbf{nf}(\varphi_2)) \\ &= size(\mathbf{nf}(\varphi_1)) + 1 + size(\mathbf{nf}(\varphi_2)) \\ &\leqslant 2^{size(\varphi_1)} + 1 + 2^{size(\varphi_2)} \\ &< 2^{size(\varphi_1)+1+size(\varphi_2)} \\ &= 2^{size(\varphi_1 \vee \varphi_2)}. \end{aligned}$$

— $\varphi_1 \wedge \varphi_2$:

This is similar to the previous case.

— $\mathsf{F}\varphi_0$:

$$\begin{aligned} size(\mathbf{nf}(\mathsf{F}\varphi_0)) &= size(tt\,\mathsf{U}\,(\mathbf{nf}(\varphi_0))) \\ &= 2 + size(\mathbf{nf}(\varphi_0)) \\ &\leqslant 2 + 2^{size(\varphi_0)} \\ &\leqslant 2^{size(\varphi_0)+1} \\ &= 2^{size(\mathsf{F}\varphi_0)}. \end{aligned}$$

— $\mathsf{G}\varphi_0$:

This case is divided into the following subcases according to the structure of φ_0:

– α:

$$\begin{aligned} size(\mathbf{nf}(\mathsf{G}\alpha)) &= size(\mathsf{G}\alpha) \\ &= 2 \\ &< 2^2 \\ &= 2^{size(\mathsf{G}\alpha)}. \end{aligned}$$

In the remaining cases we assume that $\varphi_0 \notin \mathrm{LTL}()$.

https://doi.org/10.1017/S0960129511000399 Published online by Cambridge University Press

– $\varphi_1 \wedge \varphi_2$:

$$\begin{aligned} size(\mathbf{nf}(\mathsf{G}(\varphi_1 \wedge \varphi_2))) &= size(\mathbf{nf}(\mathsf{G}\varphi_1) \wedge \mathbf{nf}(\mathsf{G}\varphi_2)) \\ &= size(\mathbf{nf}(\mathsf{G}\varphi_1)) + 1 + size(\mathbf{nf}(\mathsf{G}\varphi_2))) \\ &\leqslant 2^{size(\mathsf{G}\varphi_1)} + 1 + 2^{size(\mathsf{G}\varphi_2)} \\ &= 2^{size(\varphi_1)+1} + 1 + 2^{size(\varphi_2)+1} \\ &< 2^{size(\varphi_1)+2+size(\varphi_2)} \\ &= 2^{size(\mathsf{G}(\varphi_1 \wedge \varphi_2))}. \end{aligned}$$

– $\mathsf{F}\varphi_1$:

This case is again divided into the following subcases according to the structure of φ_1:

- α:

$$\begin{aligned} size(\mathbf{nf}(\mathsf{GF}\alpha)) &= size(\mathsf{GF}\alpha) \\ &= 3 \\ &< 2^3 \\ &= 2^{size(\mathsf{GF}\alpha)}. \end{aligned}$$

In the remaining cases we assume that $\varphi_1 \notin$ LTL().

- $\varphi_3 \vee \varphi_4$:

$$\begin{aligned} size(\mathbf{nf}(\mathsf{GF}(\varphi_3 \vee \varphi_4))) &= size(\mathbf{nf}(\mathsf{GF}\varphi_3) \vee \mathbf{nf}(\mathsf{GF}\varphi_4)) \\ &= size(\mathbf{nf}(\mathsf{GF}\varphi_3)) + 1 + size(\mathbf{nf}(\mathsf{GF}\varphi_4)) \\ &\leqslant 2^{size(\mathsf{GF}\varphi_3)} + 1 + 2^{size(\mathsf{GF}\varphi_4)} \\ &= 2^{2+size(\varphi_3)} + 1 + 2^{2+size(\varphi_4)} \\ &< 2^{size(\varphi_3)+3+size(\varphi_4)} \\ &= 2^{size(\mathsf{GF}(\varphi_3 \vee \varphi_4))}. \end{aligned}$$

- $\varphi_3 \wedge \varphi_4$:

We proceed according to the structure of φ_4:

- $\varphi_5 \vee \varphi_6$:

$$\begin{aligned} size(\mathbf{nf}(\mathsf{GF}(\varphi_3 \wedge (\varphi_5 \vee \varphi_6)))) &= size(\mathbf{nf}(\mathsf{GF}(\varphi_3 \wedge \varphi_5)) \vee \mathbf{nf}(\mathsf{GF}(\varphi_3 \wedge \varphi_6))) \\ &= size(\mathbf{nf}(\mathsf{GF}(\varphi_3 \wedge \varphi_5))) + 1 + \\ &\quad size(\mathbf{nf}(\mathsf{GF}(\varphi_3 \wedge \varphi_6))) \\ &\leqslant 2^{size(\mathsf{GF}(\varphi_3 \wedge \varphi_5))} + 1 + 2^{size(\mathsf{GF}(\varphi_3 \wedge \varphi_6))} \\ &= 2^{3+size(\varphi_3)+size(\varphi_5)} + 1 + \\ &\quad 2^{3+size(\varphi_3)+size(\varphi_6)} \\ &< 2^{4+size(\varphi_3)+size(\varphi_5)+size(\varphi_6)} \\ &= 2^{size(\mathsf{GF}(\varphi_3 \wedge (\varphi_5 \vee \varphi_6)))}. \end{aligned}$$

https://doi.org/10.1017/S0960129511000399 Published online by Cambridge University Press

- $\mathsf{F}\varphi_5$:

$$\begin{aligned} size(\mathbf{nf}(\mathsf{GF}(\varphi_3 \wedge \mathsf{F}\varphi_5))) &= size(\mathbf{nf}(\mathsf{GF}\varphi_3) \wedge \mathbf{nf}(\mathsf{GF}\varphi_5)) \\ &= size(\mathbf{nf}(\mathsf{GF}\varphi_3)) + 1 + size(\mathbf{nf}(\mathsf{GF}\varphi_5)) \\ &\leqslant 2^{size(\mathsf{GF}\varphi_3)} + 1 + 2^{size(\mathsf{GF}\varphi_5)} \\ &= 2^{2+size(\varphi_3)} + 1 + 2^{2+size(\varphi_5)} \\ &< 2^{4+size(\varphi_3)+size(\varphi_5)} \\ &= 2^{size(\mathsf{GF}(\varphi_3 \wedge \mathsf{F}\varphi_5))}. \end{aligned}$$

- $\mathsf{G}\varphi_5$:

$$\begin{aligned} size(\mathbf{nf}(\mathsf{GF}(\varphi_3 \wedge \mathsf{G}\varphi_5))) &= size(\mathbf{nf}(\mathsf{GF}\varphi_3) \wedge tt\,\mathsf{U}\,(\mathbf{nf}(\mathsf{G}\varphi_5))) \\ &= size(\mathbf{nf}(\mathsf{GF}\varphi_3)) + 3 + size(\mathbf{nf}(\mathsf{G}\varphi_5)) \\ &\leqslant 2^{size(\mathsf{GF}\varphi_3)} + 3 + 2^{size(\mathsf{G}\varphi_5)} \\ &= 2^{2+size(\varphi_3)} + 3 + 2^{1+size(\varphi_5)} \\ &< 2^{4+size(\varphi_3)+size(\varphi_5)} \\ &= 2^{size(\mathsf{GF}(\varphi_3 \wedge \mathsf{G}\varphi_5))}. \end{aligned}$$

- $\mathsf{F}\varphi_3$:

$$\begin{aligned} size(\mathbf{nf}(\mathsf{GFF}\varphi_3)) &= size(\mathbf{nf}(\mathsf{GF}\varphi_3)) \\ &\leqslant 2^{size(\mathsf{GF}\varphi_3)} \\ &= 2^{2+size(\varphi_3)} \\ &< 2^{3+size(\varphi_3)} \\ &= 2^{size(\mathsf{GFF}\varphi_3)}. \end{aligned}$$

- $\mathsf{G}\varphi_3$:

$$\begin{aligned} size(\mathbf{nf}(\mathsf{GFG}\varphi_3)) &= size(tt\,\mathsf{U}\,(\mathbf{nf}(\mathsf{G}\varphi_3))) \\ &= 2 + size(\mathbf{nf}(\mathsf{G}\varphi_3)) \\ &\leqslant 2 + 2^{size(\mathsf{G}\varphi_3)} \\ &= 2 + 2^{1+size(\varphi_3)} \\ &< 2^{3+size(\varphi_3)} \\ &= 2^{size(\mathsf{GFG}\varphi_3)}. \end{aligned}$$

https://doi.org/10.1017/S0960129511000399 Published online by Cambridge University Press

– $\varphi_1 \vee \varphi_2$:

We proceed according to the structure of φ_2:

- $\varphi_3 \wedge \varphi_4$:

$$\begin{aligned} size(\mathbf{nf}(\mathsf{G}(\varphi_1 \vee (\varphi_3 \wedge \varphi_4)))) &= size(\mathbf{nf}(\mathsf{G}(\varphi_1 \vee \varphi_3)) \wedge \mathbf{nf}(\mathsf{G}(\varphi_1 \vee \varphi_4))) \\ &= size(\mathbf{nf}(\mathsf{G}(\varphi_1 \vee \varphi_3))) + 1 + \\ &\quad size(\mathbf{nf}(\mathsf{G}(\varphi_1 \vee \varphi_4))) \\ &\leqslant 2^{size(\mathsf{G}(\varphi_1 \vee \varphi_3))} + 1 + 2^{size(\mathsf{G}(\varphi_1 \vee \varphi_4))} \\ &= 2^{2+size(\varphi_1)+size(\varphi_3)} + 1 + 2^{2+size(\varphi_1)+size(\varphi_4)} \\ &= 2^{3+size(\varphi_1)+size(\varphi_3)+size(\varphi_4)} \\ &= 2^{size(\mathsf{G}(\varphi_1 \vee (\varphi_3 \wedge \varphi_4)))}. \end{aligned}$$

- $\mathsf{F}\varphi_3$:

$$\begin{aligned} size(\mathbf{nf}(\mathsf{G}(\varphi_1 \vee \mathsf{F}\varphi_3))) &= size(\mathbf{nf}(\mathsf{G}\varphi_1) \vee tt\,\mathsf{U}\,(\mathbf{nf}(\varphi_3) \wedge \mathsf{X}(\mathbf{nf}(\mathsf{G}\varphi_1))) \vee \\ &\qquad \mathbf{nf}(\mathsf{GF}\varphi_3)) \\ &= 6 + size(\mathbf{nf}(\mathsf{G}\varphi_1)) + size(\mathbf{nf}(\varphi_3)) + \\ &\quad size(\mathbf{nf}(\mathsf{G}\varphi_1)) + size(\mathbf{nf}(\mathsf{GF}\varphi_3)) \\ &\leqslant 6 + 2^{size(\mathsf{G}\varphi_1)} + 2^{size(\varphi_3)} + 2^{size(\mathsf{G}\varphi_1)} + 2^{size(\mathsf{GF}\varphi_3)} \\ &= 6 + 2 * 2^{1+size(\varphi_1)} + 2^{size(\varphi_3)} + 2^{2+size(\varphi_3)} \\ &= 6 + 2^{2+size(\varphi_1)} + 2^{size(\varphi_3)} + 2^{2+size(\varphi_3)} \\ &< 4 * 2^{1+size(\varphi_1)+size(\varphi_3)} \\ &= 2^{3+size(\varphi_1)+size(\varphi_3)} \\ &= 2^{size(\mathsf{G}(\varphi_1 \vee \mathsf{F}\varphi_3))}. \end{aligned}$$

- $\mathsf{G}\varphi_3$:

In this case we just consider the following two structures of the whole subformula:

- $\bigvee_{\varphi' \in G} \mathsf{G}\varphi'$:

$$\begin{aligned} size\left(\mathbf{nf}\left(\mathsf{G}\left(\bigvee_{\varphi' \in G} \mathsf{G}\varphi'\right)\right)\right) &= size\left(\bigvee_{\varphi' \in G} \mathbf{nf}(\mathsf{G}\varphi')\right) \\ &= |G| - 1 + \sum_{\varphi' \in G} size(\mathbf{nf}(\mathsf{G}\varphi')) \\ &\leqslant |G| - 1 + \sum_{\varphi' \in G} 2^{size(\mathsf{G}\varphi')} \\ &< 2^{1+|G|-1+\sum_{\varphi' \in G} size(\mathsf{G}\varphi')} \\ &= 2^{size\left(\mathsf{G}\left(\bigvee_{\varphi' \in G} \mathsf{G}\varphi'\right)\right)}. \end{aligned}$$

https://doi.org/10.1017/S0960129511000399 Published online by Cambridge University Press

- $\alpha \vee \bigvee_{\varphi' \in G} \mathsf{G}\varphi'$:

$$size\left(\mathbf{nf}\left(\mathsf{G}\left(\alpha \vee \bigvee_{\varphi' \in G} \mathsf{G}\varphi'\right)\right)\right)$$

$$= size\left((\mathsf{G}\alpha) \vee \left(\alpha\,\mathsf{U}\left(\bigvee_{\varphi' \in G} \mathbf{nf}\left(\mathsf{G}\varphi'\right)\right)\right)\right)$$

$$= 5 + |G| - 1 + \sum_{\varphi' \in G} size(\mathbf{nf}(\mathsf{G}\varphi'))$$

$$\leqslant 5 + |G| - 1 + \sum_{\varphi' \in G} 2^{size(\mathsf{G}\varphi')}$$

$$< 2^{3+|G|-1+\sum_{\varphi' \in G} size(\mathsf{G}\varphi')}$$

$$= 2^{size\left(\mathsf{G}\left(\alpha \vee \bigvee_{\varphi' \in G} \mathsf{G}\varphi'\right)\right)}.$$

– $\mathsf{G}\varphi_1$:

$$size(\mathbf{nf}(\mathsf{GG}\varphi_1)) = size(\mathbf{nf}(\mathsf{G}\varphi_1))$$

$$\leqslant 2^{size(\mathsf{G}\varphi_1)}$$

$$< 2^{size(\mathsf{GG}\varphi_1)}.$$

□

The definition of the normal form and Lemma 11 directly imply the same result for general LIO formulae.

Corollary 12. For every LIO formula φ, we can effectively construct an equivalent nLIO formula $\mathbf{nf}(\varphi)$ such that $size(\mathbf{nf}(\varphi)) \leqslant 2^{size(\varphi)}$.

Lemma 13. Given an nLIO formula φ, we can effectively construct an ALBA automaton A such that $L(\varphi) = L(A)$ and the number of states of A is at most $2^{size(\varphi)}$.

Proof. Let φ be an nLIO formula and T_φ be the transition system constructed from φ. We use $|T_\varphi|$ to denote the number of its states. From Remark 5, it is easy to see that states of T_φ are sets of subformulae of φ.

Furthermore, some subformulae of φ cannot appear in these sets, for example, strict subformulae of α or $\mathsf{G}\alpha$ or $\mathsf{GF}\alpha$ formulae. Let g and f denote the number of subformulae of φ of the form $\mathsf{G}\alpha$ and $\mathsf{GF}\alpha$, respectively. We get that at most $size(\varphi) - g - 2f$ different subformulae of φ can appear in states of T_φ. Hence,

$$|T_\varphi| \leqslant 2^{size(\varphi)-g-2f}.$$

If $f = 0$, we are done as the transformation of T_φ to A_φ does not change the number of states. Thus, the automaton has at most $2^{size(\varphi)-g} \leqslant 2^{size(\varphi)}$ states.

Now assume that $f > 0$. The transformation of T_φ to A_φ replaces every state

$$S \subseteq \{\mathsf{G}\alpha, \mathsf{GF}\alpha \mid \alpha \in \mathrm{LTL}()\}$$

of T_φ by a strongly connected component with at most 2^f states (this size estimation holds even for the strongly connected components of the type shown in Figure 5). Each

https://doi.org/10.1017/S0960129511000399 Published online by Cambridge University Press

T_φ has at most 2^{g+f} such terminal components. Hence, the transformation of T_φ to A_φ adds at most

$$2^{g+f} \cdot 2^f = 2^{g+2f}$$

states. In total, the automaton A_φ has at most

$$2^{size(\varphi)-g-2f} + 2^{g+2f}$$

states. But $f > 0$ and $size(\varphi) > g + 2f$ implies

$$2^{size(\varphi)-g-2f} + 2^{g+2f} \leqslant 2^{size(\varphi)},$$

so we are done. □

The following theorem follows directly from Corollary 12 and Lemma 13.

Theorem 14. Given an LIO formula φ, we can effectively construct an ALBA automaton A such that $L(\varphi) = L(A)$ and the number of states of A is at most $2^{2^{size(\varphi)}}$.

Finally, we present a parametric formula showing that the double exponential upper bound given in the previous corollary is tight up to a constant factor.

Lemma 15. For every $n \geqslant 1$, let φ_n be an LIO formula

$$\mathsf{G}\left(\alpha \vee \bigvee_{i=1}^{n} (\mathsf{G}\beta_i \wedge \mathsf{F}\gamma_i)\right),$$

where $\alpha, \beta_i, \gamma_i \in \mathrm{LTL}()$ for $1 \leqslant i \leqslant n$. The number of states of A_{φ_n} is at least 2^{2^n}, while $size(\varphi_n) = 2 + 6 * n$.

Proof. During the transformation to normal form, we first apply the rule

$$\mathsf{G}(\varphi_1 \vee (\varphi_3 \wedge \varphi_4)) \equiv \mathsf{G}(\varphi_1 \vee \varphi_3) \wedge \mathsf{G}(\varphi_1 \vee \varphi_4)$$

and obtain an equivalent formula

$$\varphi'_n = \bigwedge_{\mathcal{I} \subseteq \{1,\ldots,n\}} \mathsf{G}\left(\alpha \vee \bigvee_{i \in \mathcal{I}} \mathsf{G}\beta_i \vee \bigvee_{i \notin \mathcal{I}} \mathsf{F}\gamma_i\right),$$

which consists of 2^n (mutually different) conjuncts. Then each of the conjuncts is transformed using the rule

$$\mathsf{G}(\varphi_1 \vee \mathsf{F}\varphi_3) \equiv \mathsf{G}\varphi_1 \vee tt\,\mathsf{U}\,(\varphi_3 \wedge \mathsf{X}(\mathsf{G}\varphi_1)) \vee \mathsf{GF}\varphi_3,$$

and then, finally, using the rule for

$$\mathsf{G}\left(\alpha \vee \bigvee_{\varphi' \in G} \mathsf{G}\varphi'\right).$$

https://doi.org/10.1017/S0960129511000399 Published online by Cambridge University Press

This transforms every conjunct into a long disjunction. The resulting normal form formula is of the form

$$\mathbf{nf}(\varphi_n) = \bigwedge_{\mathscr{I} \subseteq \{1,\ldots,n\}} \left(\mathsf{G}\alpha \vee \bigvee_{i \in \mathscr{I}} \mathsf{G}\beta_i \vee \bigvee_{i \notin \mathscr{I}} \mathsf{GF}\gamma_i \vee \Phi_{\mathscr{I}} \vee \ldots \right),$$

where $\Phi_{\mathscr{I}}$ is a unique subformula of $\mathbf{nf}(\varphi_n)$ for every $\mathscr{I} \subseteq \{1,\ldots,n\}$. The formula $\Phi_{\mathscr{I}}$ is obtained as a

$$tt\,\mathsf{U}\,(\varphi_3 \wedge \mathsf{X}(\mathsf{G}\varphi_1))$$

part of the transformation rule for $\mathsf{G}(\varphi_1 \vee \mathsf{F}\varphi_3)$. (For $\mathscr{I} = \{1,\ldots,n\}$, there is no F operator in the conjunct, so the transformation rule is not used at all. Therefore, we set $\Phi_{\{1,\ldots,n\}}$ equal to

$$\alpha\,\mathsf{U}\left(\bigvee_{i \in \{1,\ldots,n\}} \mathsf{G}\beta_i \right),$$

which is obtained by the transformation rule for

$$\mathsf{G}\left(\alpha \vee \bigvee_{\varphi' \in G} \mathsf{G}\varphi' \right)$$

and is also a unique subformula of $\mathbf{nf}(\varphi_n)$.)

The transformation into ALBA continues by the construction of sets $R(.)$. For every $\mathscr{I} \subseteq \{1,\ldots,n\}$, there is a formula $\alpha'_{\mathscr{I}} \in \mathrm{LTL}()$ such that we add (except for others) an item $(\alpha'_{\mathscr{I}}, \{\Phi_{\mathscr{I}}\})$ into the set $R(.)$ of the conjunct induced by $\mathscr{I}$. Hence, the set $R(\mathbf{nf}(\varphi_n))$ generates at least 2^{2^n} pairs with unique second element. Hence, the ALBA automaton for φ contains at least 2^{2^n} states. □

5. Implementation and experimental results

We decided to implement the translation presented in the previous section in order to answer the following three questions:

(1) Can our translation be applied to a substantial proportion of the formulae actually used in verification practice?
(2) Are the ALBA automata produced by our translation comparable (in the sense of their size) with the automata produced by standard LTL to BA translations?
(3) Are the resources (time and memory) needed for our translation comparable to the resources needed for standard translations?

We compare our LIO to ALBA translation with the LTL to BA translation introduced in Gastin and Oddoux (2001). This LTL to BA translation uses alternating co-Büchi automata and generalised Büchi automata as intermediate formalisms and is one of the best known LTL to BA translation algorithms: it is fast and the resulting automata are small. To be precise, we compare our implementation of LIO to ALBA translation with two implementations of the LTL to BA translation: the original implementation `ltl2ba` (Gastin and Oddoux 2001) and `ltl2ba-divine`, which is the implementation

https://doi.org/10.1017/S0960129511000399 Published online by Cambridge University Press

employed in the distributed model checker DiVinE (Barnat *et al.* 2006). Although the two implementations use the same algorithm, they do not always give the same results. The reason is that `ltl2ba` uses some on-the-fly optimisations that do not work in the same way in `ltl2ba-divine`, and `ltl2ba-divine` also applies post-optimisations, which are described in Etessami and Holzmann (2000), whereas `ltl2ba` does not. Our implementation of the LIO to ALBA translation uses some parts of `ltl2ba-divine`, in particular, the pre- and post-optimisations.

As we are interested in the practical relevance of LIO to ALBA translation, we do not evaluate the translation on any randomly generated formulae. We simply use publicly available specification formulae from two different sources:

— Spec Patterns (Dwyer *et al.* 1998), which contains 55 LTL formulae and are available online[†] – we refer to these formulae as $\varphi_1, \varphi_2, \ldots, \varphi_{55}$.
— BEEM (benchmark for explicit model checkers) (Pelánek 2007), which contains the following 20 LTL formulae:

$$\begin{aligned}
\psi_1 &= \mathsf{G}(a \to \mathsf{F}b) & \psi_{11} &= \neg(\neg(a \vee b)\,\mathsf{U}\,c) \wedge \mathsf{G}(d \to \neg(\neg(a \vee b)\,\mathsf{U}\,c)) \\
\psi_2 &= ((\mathsf{GF}a) \wedge (\mathsf{GF}b)) \to (\mathsf{GF}c) & \psi_{12} &= (\mathsf{G}\neg a) \to (\mathsf{G}\neg b) \\
\psi_3 &= \mathsf{G}(a \to (b \wedge (c\,\mathsf{U}\,d))) & \psi_{13} &= \mathsf{G}(a \to ((\mathsf{G}\neg b) \vee (\neg c\,\mathsf{U}\,b))) \\
\psi_4 &= \mathsf{F}(a \vee b) & \psi_{14} &= \mathsf{G}(a \to (b\,\mathsf{R}\,(\neg c \vee b))) \\
\psi_5 &= \mathsf{GF}(a \vee b) & \psi_{15} &= \mathsf{G}((a \wedge b) \to (\neg b\,\mathsf{R}\,(a \vee \neg b))) \\
\psi_6 &= (a\,\mathsf{U}\,b) \to ((c\,\mathsf{U}\,d) \vee \mathsf{G}c) & \psi_{16} &= \mathsf{G}(a \to (\mathsf{F}(b \wedge c))) \\
\psi_7 &= \mathsf{G}(a \to (\neg b\,\mathsf{U}\,(b\,\mathsf{U}\,(b \wedge c)))) & \psi_{17} &= \mathsf{G}(a \to (\neg b\,\mathsf{U}\,(b\,\mathsf{U}\,(\neg b \wedge (c\,\mathsf{R}\,\neg b))))) \\
\psi_8 &= \mathsf{G}(a \to (b\,\mathsf{R}\,\neg c)) & \psi_{18} &= \mathsf{G}(a \to (\neg b\,\mathsf{U}\,(b\,\mathsf{U}\,(\neg b\,\mathsf{U}\,(b\,\mathsf{U}\,(b \wedge c)))))) \\
\psi_9 &= \mathsf{G}(\neg a \to \mathsf{F}a) & \psi_{19} &= (\mathsf{GF}a) \to (\mathsf{GF}b) \\
\psi_{10} &= \mathsf{G}(a \to \mathsf{F}(b \vee c)) & \psi_{20} &= \mathsf{GF}(a \vee b) \wedge \mathsf{GF}(c \vee b)\,.
\end{aligned}$$

Note that we do not translate the specification formulae themselves, but their negations since model-checking algorithms usually need automata representing behaviours violating the specification.

A careful manual analysis shows that the negations of 49 out of the 55 formulae from Spec Patterns can be translated into ALBA automata (and hence these negations can be expressed in LIO). However, only 10 of these negations lie syntactically within LIO (this is due to the fact that negation can only appear in a LIO formula in subformulae of LTL(F, G)). Similarly, only 14 of the negations of the 20 BEEM formulae lie syntactically within LIO.

To increase the number of potential input formulae for the LIO to ALBA translation, we have extended the syntax of LIO with the temporal operators R and W, as we mentioned in Subsection 2.3. Furthermore, we employ the following equivalences to rewrite a non-LIO formula into an equivalent LIO formula:

$$\neg(\varphi\,\mathsf{W}\,\psi) \equiv \neg\psi\,\mathsf{U}\,(\neg\varphi \wedge \neg\psi) \qquad \neg(\varphi\,\mathsf{U}\,\psi) \equiv (\mathsf{G}\neg\psi) \vee (\neg\psi\,\mathsf{U}\,(\neg\varphi \wedge \neg\psi))$$

[†] `http://patterns.projects.cis.ksu.edu/documentation/patterns/ltl.shtml`

https://doi.org/10.1017/S0960129511000399 Published online by Cambridge University Press

$$\begin{aligned}
\varphi\,\mathsf{U}\,(\mathsf{F}\psi) &\equiv \mathsf{F}\psi & \neg(\varphi\,\mathsf{U}\,(\mathsf{F}\psi)) &\equiv \mathsf{G}\neg\psi\\
\varphi\,\mathsf{U}\,(\mathsf{G}\psi) &\equiv \mathsf{FG}\psi \wedge \mathsf{G}(\varphi \vee \mathsf{G}\psi) & \neg(\varphi\,\mathsf{U}\,(\mathsf{G}\psi)) &\equiv \mathsf{GF}\neg\psi \vee \mathsf{F}(\neg\varphi \wedge \mathsf{F}\neg\psi)\\
\varphi\,\mathsf{W}\,(\mathsf{F}\psi) &\equiv \mathsf{G}\varphi \vee \mathsf{F}\psi & \neg(\varphi\,\mathsf{W}\,(\mathsf{F}\psi)) &\equiv \mathsf{F}\neg\varphi \wedge \mathsf{G}\neg\psi\\
\varphi\,\mathsf{W}\,(\mathsf{G}\psi) &\equiv \mathsf{G}(\varphi \vee \mathsf{G}\psi) & \neg(\varphi\,\mathsf{W}\,(\mathsf{G}\psi)) &\equiv \mathsf{F}(\neg\varphi \wedge \mathsf{F}\neg\psi)
\end{aligned}$$

$$\begin{aligned}
(\mathsf{F}\varphi)\,\mathsf{U}\,\psi &\equiv \psi \vee \mathsf{F}(\mathsf{X}\psi \wedge \mathsf{F}\varphi) & \neg((\mathsf{F}\varphi)\,\mathsf{U}\,\psi) &\equiv \neg\psi \wedge \mathsf{G}(\mathsf{X}\neg\psi \vee \mathsf{G}\neg\varphi)\\
(\mathsf{G}\varphi)\,\mathsf{U}\,\psi &\equiv \psi \vee (\mathsf{G}\varphi \wedge \mathsf{F}\psi) & \neg((\mathsf{G}\varphi)\,\mathsf{U}\,\psi) &\equiv \mathsf{G}\neg\psi \vee (\mathsf{F}\neg\varphi \wedge \neg\psi)\\
(\mathsf{F}\varphi)\,\mathsf{W}\,\psi &\equiv \mathsf{GF}\varphi \vee \mathsf{F}(\mathsf{X}\psi \wedge \mathsf{F}\varphi) & \neg((\mathsf{F}\varphi)\,\mathsf{W}\,\psi) &\equiv \mathsf{FG}\neg\varphi \wedge \mathsf{G}(\mathsf{X}\neg\psi \vee \mathsf{G}\neg\varphi)\\
(\mathsf{G}\varphi)\,\mathsf{W}\,\psi &\equiv \psi \vee \mathsf{G}\varphi & \neg((\mathsf{G}\varphi)\,\mathsf{W}\,\psi) &\equiv \neg\psi \wedge \mathsf{F}\neg\varphi
\end{aligned}$$

$$\begin{aligned}
\mathsf{G}(\varphi \vee \mathsf{XG}\psi) &\equiv \mathsf{G}\varphi \vee (\varphi\,\mathsf{U}\,\mathsf{XG}\psi)\\
\mathsf{G}(\varphi \vee \mathsf{XF}\psi) &\equiv \mathsf{G}\varphi \vee \mathsf{XF}(\psi \wedge \mathsf{G}\varphi) \vee \mathsf{GF}\psi\\
\mathsf{GF}(\varphi \wedge \mathsf{XG}\psi) &\equiv \mathsf{GF}\varphi \wedge \mathsf{FG}\psi\\
\text{if } \varphi \to \psi \text{ then } \mathsf{G}(\varphi\,\mathsf{U}\,\psi) &\equiv \mathsf{G}\varphi \wedge \mathsf{GF}\psi\\
\varphi\,\mathsf{U}\,(\mathsf{G}\psi) &\equiv \mathsf{G}(\varphi\,\mathsf{U}\,(\mathsf{G}\psi))\\
\varphi\,\mathsf{W}\,(\mathsf{G}\psi) &\equiv \mathsf{G}(\varphi\,\mathsf{W}\,(\mathsf{G}\psi))
\end{aligned}$$

Using these equivalences, we can automatically translate the negations of 33 out of the 55 formulae from Spec Patterns and the negations of 17 out of the 20 formulae from BEEM. Hence, it seems that the answer to our first question is yes.

To answer the other two questions, we executed the three implementations on the negations of 33 Spec Patterns formulae and 17 BEEM formulae mentioned above. The implementations were executed with all available optimisations in order to get the smallest automata. It is worth mentioning that all the optimisations applied in `lio2alba` preserve the ALBA form of the automata.

The results are presented in Table 1 (for the negations of Spec Patterns formulae) and Table 2 (for the negations of BEEM formulae). For each formula and each implementation, the tables contain the number of states (st.) and transitions (tr.) of the resulting automaton and the memory (mem.) and time needed for the translation. In counting the number of transitions, all transitions $p \xrightarrow{e} q$ for a fixed p, q are counted as one transition. The memory is measured in kB and the time is in seconds (or in minutes when indicated by 'm').

All computations were done on a server with 8 processors Intel® Xeon® X7560, 448 GiB RAM and a 64-bit version of GNU/Linux (kernel version 2.6.32). To measure the time needed for computation, we used a built-in system program `time`. To measure the peak memory consumption, we used the program `tstime`[†].

The tables show that for most of the inputs considered, all three implementations produce automata with the same number of states and transitions. The inputs where this is not true are marked with '■'. For some of them, the size of the automaton produced by `lio2alba` coincides with the smaller of the other two automata. In some cases, the ALBA produced by `lio2alba` is slightly bigger, but the difference in sizes is not dramatic.

[†] `tstime` is available at `http://bitbucket.org/gsauthof/tstime/overview/`.

https://doi.org/10.1017/S0960129511000399 Published online by Cambridge University Press

Table 1. *Results for negations of Spec Patterns formulae.*

	`lio2alba`				`ltl2ba-divine`				`ltl2ba`			
φ	st.	tr.	mem.	time	st.	tr.	mem.	time	st.	tr.	mem.	time
φ_1	2	3	1280	0.006	2	3	1272	0.009	2	3	592	0.004
φ_2	3	5	1308	0.007	3	5	1272	0.008	3	5	616	0.005
φ_3	3	6	1304	0.006	3	6	1276	0.050	3	6	608	0.005
φ_4	4	8	1320	0.007	4	8	1284	0.006	4	8	632	0.005
φ_5	3	6	1304	0.007	3	6	1288	0.007	3	6	624	0.005
φ_6	1	1	1272	0.007	1	1	1276	0.006	1	1	580	0.005
φ_7	2	3	1288	0.007	2	3	1272	0.006	2	3	620	0.004
φ_8	3	6	1320	0.007	3	6	1280	0.006	3	6	620	0.005
φ_9	3	5	1304	0.007	3	5	1284	0.007	3	5	624	0.005
■ φ_{10}	**4**	**8**	**1304**	**0.006**	**3**	**5**	**1280**	**0.006**	**3**	**5**	**620**	**0.005**
φ_{11}	6	11	1332	0.008	6	11	1324	0.012	6	11	640	0.005
φ_{16}	2	3	1284	0.006	2	3	1268	0.006	2	3	600	0.004
φ_{17}	3	5	1312	0.007	3	5	1276	0.006	3	5	612	0.005
φ_{18}	3	6	1300	0.006	3	6	1280	0.006	3	6	616	0.005
φ_{19}	4	8	1320	0.007	4	8	1280	0.006	4	8	632	0.005
φ_{20}	3	6	1300	0.007	3	6	1284	0.006	3	6	620	0.006
φ_{21}	2	3	1292	0.006	2	3	1276	0.006	2	3	608	0.005
φ_{22}	3	5	1316	0.007	3	5	1272	0.006	3	5	624	0.005
φ_{24}	4	8	1332	0.008	4	8	1288	0.007	4	8	632	0.005
φ_{25}	3	6	1308	0.007	3	6	1288	0.007	3	6	636	0.005
φ_{26}	2	3	1296	0.006	2	3	1272	0.006	2	3	616	0.006
φ_{27}	3	5	1332	0.008	3	5	1280	0.006	3	5	628	0.005
φ_{28}	3	6	1312	0.006	3	6	1276	0.006	3	6	624	0.005
φ_{29}	4	7	1348	0.009	4	8	1292	0.007	4	8	660	0.005
■ φ_{30}	**5**	**12**	**1328**	**0.008**	**4**	**8**	**1300**	**0.007**	**4**	**8**	**660**	**0.005**
■ φ_{36}	**3**	**5**	**1324**	**0.008**	**3**	**5**	**1284**	**0.006**	**4**	**8**	**620**	**0.005**
φ_{37}	4	7	1340	0.008	4	7	1284	0.008	4	7	656	0.005
■ φ_{39}	**4**	**7**	**1344**	**0.008**	**4**	**7**	**1292**	**0.007**	**5**	**10**	**664**	**0.005**
■ φ_{40}	**4**	**8**	**1372**	**0.011**	**4**	**8**	**1324**	**0.008**	**14**	**43**	**668**	**0.006**
φ_{41}	4	8	1324	0.006	4	8	1288	0.007	4	8	632	0.004
■ φ_{46}	**4**	**8**	**1324**	**0.006**	**3**	**6**	**1280**	**0.006**	**3**	**6**	**624**	**0.005**
φ_{48}	4	9	1320	0.007	4	9	1280	0.007	4	9	636	0.005
■ φ_{53}	**6**	**14**	**1324**	**0.008**	**4**	**10**	**1292**	**0.007**	**4**	**10**	**636**	**0.005**

https://doi.org/10.1017/S0960129511000399 Published online by Cambridge University Press

Table 2. *Results for negations of BEEM formulae.*

	`lio2alba`				`ltl2ba-divine`				`ltl2ba`			
ψ	st.	tr.	mem.	time	st.	tr.	mem.	time	st.	tr.	mem.	time
ψ_1	2	3	1300	0.006	2	3	1272	0.006	2	3	612	0.004
ψ_2	4	10	1308	0.006	4	11	1300	0.007	4	10	624	0.005
■ ψ_3	**4**	**8**	**1316**	**0.007**	**3**	**6**	**1284**	**0.006**	**3**	**6**	**620**	**0.005**
ψ_4	1	1	1284	0.006	1	1	1272	0.006	1	1	596	0.004
ψ_5	2	3	1300	0.006	2	3	1280	0.006	2	3	616	0.005
ψ_6	4	9	1296	0.007	4	9	1284	0.006	4	9	628	0.005
ψ_8	3	6	1296	0.006	3	6	1276	0.007	3	6	612	0.005
ψ_9	2	3	1300	0.006	2	3	1268	0.006	2	3	612	0.004
ψ_{10}	2	3	1296	0.006	2	3	1276	0.007	2	3	612	0.006
ψ_{11}	4	9	1308	0.007	4	9	1288	0.006	4	9	632	0.005
ψ_{12}	2	3	1304	0.006	2	3	1276	0.006	2	3	608	0.005
ψ_{13}	4	8	1320	0.007	4	8	1284	0.006	4	8	616	0.005
ψ_{14}	3	6	1300	0.007	3	6	1280	0.006	3	6	620	0.004
ψ_{15}	3	5	1300	0.006	3	5	1280	0.007	3	5	624	0.005
ψ_{16}	2	3	1300	0.006	2	3	1280	0.006	2	3	624	0.005
ψ_{19}	3	6	1300	0.006	3	6	1280	0.007	3	6	624	0.005
ψ_{20}	3	5	1308	0.006	3	5	1284	0.007	3	5	620	0.004

An analysis of the execution time of the implementations shows that `lio2alba` is fully comparable with the other two implementations (`ltl2ba` seems to be a bit faster in some cases). The memory requirements of `lio2alba` and `ltl2ba-divine` are almost the same, while `ltl2ba` requires roughly half this amount.

Since the data presented by Tables 1 and 2 might produce the feeling that the results of all the three implementations are always more or less the same, Table 3 compares the three implementations on the following three parametric formulae:

$$\theta_n = \neg((\mathsf{GF}p_1 \wedge \mathsf{GF}p_2 \wedge \ldots \wedge \mathsf{GF}p_n) \rightarrow \mathsf{G}(p \rightarrow \mathsf{F}r))$$
$$\zeta_n = \mathsf{G}((\mathsf{F}p_1 \wedge \mathsf{F}q_1) \vee (\mathsf{F}p_2 \wedge \mathsf{F}q_2) \vee \ldots \vee (\mathsf{F}p_n \wedge \mathsf{F}q_n))$$
$$\pi_n = \mathsf{G}(p_1 \vee \mathsf{F}q_1)) \vee (\mathsf{G}(p_2 \vee \mathsf{F}q_2)) \vee \ldots \vee (\mathsf{G}(p_n \vee \mathsf{F}q_n)$$

(The formula θ_n is taken from Gastin and Oddoux (2001).) Each formula illustrates a different phenomenon:

— The formulae θ_n exemplify a situation where all three implementations produce automata of essentially the same size (`ltl2ba-divine` produces automata with slightly more transitions) but with very different computational demands. More precisely, the time and memory requirements of `ltl2ba` and `ltl2ba-divine` grow very quickly

https://doi.org/10.1017/S0960129511000399 Published online by Cambridge University Press

Table 3. *Results for testing formulae.*

	`lio2alba`				`ltl2ba-divine`				`ltl2ba`			
φ	st.	tr.	mem.	time	st.	tr.	mem.	time	st.	tr.	mem.	time
θ_5	7	28	1344	0.009	7	33	2584	0.184	7	28	696	0.033
θ_6	8	36	1364	0.009	8	42	6376	1.727	8	36	744	0.220
θ_7	9	45	1380	0.011	9	52	23 044	29.991	9	45	872	1.645
θ_8	10	55	1404	0.012	10	63	95 392	8.59m	10	55	1136	19.591
θ_9	11	66	1428	0.015					11	66	1268	3.02m
θ_{10}	12	78	1448	0.016					12	78	1940	45.80m
θ_{20}	22	253	1956	0.035								
θ_{40}	42	903	4964	0.236								
θ_{80}	82	3403	25 488	4.005								
θ_{160}	162	13 203	177 080	1.70m								
θ_{320}	322	52 003	1 342 464	49.54m								
ζ_1	4	9	1296	0.007	3	8	1280	0.007	3	8	608	0.006
ζ_2	15	46	1380	0.012	35	313	1680	0.024	40	549	748	0.011
ζ_3	68	236	2096	0.136	168	2970	5764	0.284	224	9450	2040	0.082
ζ_4	346	1506	80 312	24.673	729	24 075	43 332	6.252	1152	139 239	15 060	2.495
π_2	9	20	1332	0.011	5	12	1292	0.011	5	12	624	0.008
π_3	27	76	1404	0.016	13	56	1356	0.013	13	56	640	0.009
π_4	114	457	2128	0.085	40	640	3324	0.082	40	640	936	0.020
π_5	324	1587	5048	0.404	96	3072	14 832	5.144	96	3072	2008	0.142
π_6	922	5641	21 232	3.049	224	14 336	88 940	3.34m	224	14 336	7388	1.336

compared with `lio2alba`. Note that after θ_{10}, the index of the formulae grows exponentially. An empty cell indicates that the computation did not finish within an hour. In the case of θ_n formulae, the post-optimisations available in `ltl2ba-divine` and `lio2alba` do not affect the size of the automata produced. If we switch these post-optimisations off, `ltl2ba-divine` translates θ_8 in 7.86m and θ_9 in 102.12m (which is over the one hour limit), while `lio2alba` translates θ_{320} in 1.78m and even θ_{640} in 27.74m. This shows that the post-optimisations of large automata can consume a substantial part of the computation time (for example, 96% in the case of θ_{320} translated by `lio2alba`).

— Formulae ζ_n demonstrate that `lio2alba` can produce much smaller automata than the other two implementations.

— Formulae π_n exhibit what is probably the most surprising phenomenon. The number of states of the automata produced by `lio2alba` grows faster than the number of

https://doi.org/10.1017/S0960129511000399 Published online by Cambridge University Press

states of automata generated by `ltl2ba` and `ltl2ba-divine`, while the opposite holds for the number of transitions.

6. Conclusion

In this paper we have introduced a new class of Büchi automata called *Almost linear Büchi automata* (ALBA) and an expressively equivalent fragment of LTL called *LIO*. To prove that ALBA and LIO are equivalent, we described a translation of LIO formulae into equivalent ALBA automata together with a reverse translation. We provided a double exponential upper bound on the size of ALBA automata produced by our translation from LIO formulae, and showed that the bound is tight. As standard LTL to Büchi automata translations are only exponential, whether there exists an exponential LIO to ALBA translation remains an open question.

We have implemented the LIO to ALBA translation and compared it with two implementations of a very popular translation of LTL to Büchi automata suggested by Gastin and Oddoux (Gastin and Oddoux 2001), namely the original implementation (`ltl2ba`) and the one used in DiVinE (Barnat *et al.* 2006) (`ltl2ba-divine`). For the comparison, we used negations of specification formulae from Spec Patterns (Dwyer *et al.* 1998) and BEEM (Pelánek 2007). Based on the assumption that Spec Patterns and BEEM provide a representative sample of real-life specification formulae, we can interpret the experimental results as follows:

— Our LIO to ALBA translation (with some enhancements we have described) can be applied to a substantial proportion of negated specification formulae (50 out of the 75 specification formulae considered).
— When applied to negated specification formulae, the translation, with some standard optimisations, produces ALBA automata of approximately the same sizes as the Büchi automata produced by the reference implementations.
— When applied to negated specification formulae, the time and memory consumption of our translation is fully comparable to `ltl2ba-divine`, while `ltl2ba` runs slightly faster and requires approximately half the memory.

We also presented some artificial formulae showing that the LIO to ALBA translation can sometimes outperform the reference implementations in terms of speed, memory consumption and/or the size of the automata produced. These results provide clear motivation for further improvements of standard translations of LTL to Büchi automata.

To sum up, the suggested LIO to ALBA translation can generate reasonably small ALBA automata for many negated specification formulae. The current challenge is to develop improvements of the model-checking process that profit from the specific shape of ALBA automata.

Acknowledgements

We thank the two anonymous reviewers for interesting suggestions and for providing the motivation to improve the paper.

https://doi.org/10.1017/S0960129511000399 Published online by Cambridge University Press

References

Babiak, T. (2010) Almost linear Büchi automata. Master's thesis, Faculty of Informatics, Masaryk University.

Babiak, T., Řehák, V. and Strejček, J. (2009) Almost linear Büchi automata. In: EXPRESS 2009. *Electronic Proceedings in Theoretical Computer Science* **8** 16–25.

Barnat, J., Brim, L., Černá, I., Moravec, P., Ročkai, P. and Šimeček, P. (2006) DiVinE – A Tool for Distributed Verification. In: Ball, T. and Jones, R. B. (eds.) Computer Aided Verification: Proceedings 18th International Conference, CAV 2006. *Springer-Verlag Lecture Notes in Computer Science* **4144** 278–281.

Černá, I. and Pelánek, R. (2003) Relating hierarchy of temporal properties to model checking. In: Proceedings of the 30th Symposium on Mathematical Foundations of Computer Science (MFCS'03). *Springer-Verlag Lecture Notes in Computer Science* **2747** 318–327.

Courcoubetis, C., Vardi, M. Y., Wolper, P. and Yannakakis, M. (1992) Memory-efficient algorithms for the verification of temporal properties. *Formal Methods in System Design* **1** (2/3) 275–288.

Dwyer, M. B., Avrunin, G. S. and Corbett, J. C. (1998) Property specification patterns for finite-state verification. In: *Proc. 2nd Workshop on Formal Methods in Software Practice (FMSP-98)*, ACM Press 7–15.

Etessami, K. and Holzmann, G. J. (2000) Optimizing Büchi automata. In: Palamidessi, C. (ed.) Proceedings CONCUR 2000 – Concurrency Theory, 11th International Conference. *Springer-Verlag Lecture Notes in Computer Science* **1877** 153–167.

Gastin, P. and Oddoux, D. (2001) Fast LTL to Büchi automata translation. In: Berry, G., Comon, H. and Finkel, A. (eds.) Proceedings of the 13th International Conference on Computer Aided Verification (CAV'01). *Springer-Verlag Lecture Notes in Computer Science* **2102** 53–65.

Holzmann, G., Peled, D. and Yannakakis, M. (1996) On nested depth first search. In: *The Spin Verification System, Proceedings of the Second Spin Workshop*, American Mathematical Society 23–32.

Lamport, L. (1983) What good is temporal logic? In: Mason, R. E. A. (ed.) *Proceedings of the IFIP Congress on Information Processing*, North-Holland 657–667.

Maidl, M. (2000) The common fragment of CTL and LTL. In: Young, D. C. (ed.) *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science (FOCS'00)*, IEEE Computer Society Press 643–652.

Manna, Z. and Pnueli, A. (1990) A hierarchy of temporal properties. In: *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC'90)*, ACM Press 377–410.

Pelánek, R. (2007) BEEM: Benchmarks for explicit model checkers. In: Bosnacki, D. and Edelkamp, S. (eds.) Model Checking Software, Proceedings 14th International SPIN Workshop. *Springer-Verlag Lecture Notes in Computer Science* **4595** 263–267.

Perrin, D. and Pin, J.-E. (2004) Infinite words. *Pure and Applied Mathematics* **141**.

Pnueli, A. (1977) The temporal logic of programs. In: *Proceedings 18th IEEE Symposium on the Foundations of Computer Science*, IEEE Computer Society Press 46–57.

Rozier, K. Y. and Vardi, M. Y. (2007) LTL satisfiability checking. In: Bosnacki, D. and Edelkamp, S. (eds.) Model Checking Software, Proceedings 14th International SPIN Workshop. *Springer-Verlag Lecture Notes in Computer Science* **4595** 149–167.

Strejček, J. (2004) *Linear Temporal Logic: Expressiveness and Model Checking*, Ph.D. thesis, Faculty of Informatics, Masaryk University in Brno.

Tarjan, R. E. (1972) Depth-first search and linear graph algorithms. *SIAM J. Comput.* **1** (2) 146–160.

https://doi.org/10.1017/S0960129511000399 Published online by Cambridge University Press