Almost Linear Büchi Automata

Tomáš Babiak* Vojtěch Řehák† Jan Strejček‡

Faculty of Informatics
Masaryk University
Brno, Czech Republic

{xbabiak, rehak, strejcek}@fi.muni.cz

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. While standard translations of LTL into BA use some intermediate formalisms, the presented translation of LIO into ALBA is direct. As we expect applications of ALBA in model checking, we compare the expressiveness of ALBA with other classes of Büchi automata studied in this context and we indicate possible applications.

1 Introduction

The growing number of concurrent software and/or hardware systems puts more emphasis on development of automatic verification methods applicable in practice. One of the most promising methods is LTL model checking. The main problem of this verification method is the *state explosion problem* and consequent high computational complexity. While symbolic approaches to model checking partly solve the problem for hardware systems, there is still no satisfactory solution for model checking of software systems. The most promising approach seems to be a combination of abstraction methods, reduction methods, and optimized model checking algorithms. Reduction methods and optimizations of the algorithms are often based on some specific properties of the specification formula or the model. For example, one of the most effective reduction methods called *partial order reduction* employs the fact that specification formulae usually do not use the modality *next* and thus they describe *stutter-invariant* properties [6].

We have realized that all formulae of the *restricted temporal logic* [10] (i.e. formulae using only temporal operators *eventually* and *always*) can be translated to Büchi automata that are linear (1-weak), possibly with an exception of terminal strongly connected components. These terminal components have also a specific property: they accept only infinite words over a set of letters, where some selected letters appear infinitely often. We call such automata *Almost linear Büchi automata (ALBA)*. In this paper we study mainly the expressive power of these automata.

Searching for the precise class of LTL formulae corresponding to ALBA automata results in the definition of an LTL fragment named *LIO* (the abbreviation for *linear* 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. While standard translations of LTL formulae into BA use

*The co-author has been partially supported by the Ministry of Education of the Czech Republic, project No. MSM0021622419.

†The co-author has been partially supported by the research centre "Institute for Theoretical Computer Science (ITI)", project No. 1M0545, and by the Czech Science Foundation, grant No. 201/08/P459.

‡The co-author has been partially supported by the Ministry of Education of the Czech Republic, project No. MSM0021622419, and by the Czech Science Foundation, grant No. 201/08/P375.

Preliminary Report. Final version to appear in:
EXPRESS 2009

either generalized Büchi automata [5] or alternating 1-weak Büchi automata [9, 13] as an intermediate formalism, the presented translation of LIO to ALBA works directly. Further, there exist LIO formulae such that the corresponding Büchi automata created by the mentioned standard translations are not ALBA.

Related work Some observations regarding specific structure of Büchi automata corresponding to some LTL fragments have been already published in [1]. The paper states that two classes of Manna and Pnueli's hierarchy of temporal properties [8], namely *guarantee* and *persistence* formulae, can be translated into *terminal* and *weak* automata, respectively. A Büchi automaton is *terminal*, if every accepting state has a loop transition under each letter. An automaton is weak if each strongly connected component consists either of accepting or non-accepting states. The paper also suggests some improvements of the standard model checking algorithms employing the specific structure of the considered property automata. Let us note that LIO is incomparable with both guarantee and persistence formulae.

The paper is structured as follows. Section 2 recalls the definition of LTL and introduces LIO. Various kinds of Büchi automata including almost linear BA are defined in Section 3. Translations are presented in Section 4 (ALBA $\rightarrow$ LIO) and Section 5 (LIO $\rightarrow$ ALBA). Section 6 sums up the presented results and mentions some topics for future research.

2 Linear temporal logic (LTL)

The syntax of *Linear Temporal Logic* (LTL) [11] is defined as follows

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

where tt stands for *true*, a ranges over a countable set AP of *atomic propositions*, F, G, X, and U are modal operators called *eventually*, *always*, *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 $u = u(0)u(1)u(2)\ldots \in (2^{AP'})^{\omega}$, by u_i we denote the i^{th} suffix of u, i.e. $u_i = u(i)u(i+1)\ldots$.

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 $u \models \varphi_2$
$u \models \mathsf{F}\varphi$	iff	$\exists i \geq 0 \,.\, u_i \models \varphi$
$u \models \mathsf{G}\varphi$	iff	$\forall i \geq 0 \,.\, u_i \models \varphi$
$u \models \mathsf{X}\varphi$	iff	$u_1 \models \varphi$
$u \models \varphi_1 \,\mathsf{U}\, \varphi_2$	iff	$\exists i \geq 0 \,.\, (\, u_i \models \varphi_2$ and $\forall 0 \leq j < i \,.\, u_j \models \varphi_1 \,)$

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

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

For a set $\{O_1, \ldots, O_n\}$ of modalities, LTL$(O_1, \ldots, O_n)$ denotes the LTL fragment containing all formulae with modalities $O_1, \ldots, O_n$ only. We will use mainly the fragments LTL(F, G) with modalities eventually and always and LTL() without any modalities. 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 $e \models \alpha$ iff $ew \models \alpha$ for some $w \in \Sigma^{\omega}$.

2.1 The LIO fragment

The *LIO* fragment is defined as

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

where ψ ranges over LTL(F,G) and α over LTL$()$.

The fragment does not fit into any standard taxonomy of LTL fragments (see [12]), but it is a generalization of two standard LTL fragments:

- LTL(F,G) - the fragment of all LTL formulae using operators F and G only. This fragment is also known as *restricted temporal logic* [10].
- flatLTL$^+(\mathsf{U},\mathsf{X})$ - the fragment of all flat LTL(U,X) formulae in positive form. A formula is *flat* [2] if the left subformula of each U operator is from LTL$()$. A formula is in *positive* form if there is no modal operator in the scope of any negation.

In Subsection 3.2 we show that LIO contains also all languages expressible as negations of LTL$^{\text{det}}$ formulae. The fragment LTL$^{\text{det}}$ is better known as the common fragment of CTL and LTL [7].

The LIO fragment covers many specification formulae 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 needs to be translated into Büchi automata.

3 Büchi automata (BA)

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

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

We usually write $p \overset{e}{\to} q$ 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 $p \overset{e}{\to} q$. An edge $p \overset{e}{\to} p$ 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

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

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 $p \overset{*}{\to} q$, if $p = q$ or there exists a sequence

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

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 $p \xrightarrow{*} q$ 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 been considered in the context of model checking so far. A Büchi automaton $(\Sigma, Q, q_0, \delta, F)$ is called

- *terminal* if for each $p \in F$ and $a \in \Sigma$ it holds that $\delta(p,a) \neq \emptyset$ and $\delta(p,a) \subseteq F$,
- *weak* if every SCC of the automaton 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.

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

Given an automaton A and its state q, by A_q we denote the automaton A where the initial state is changed to q. Further, a strongly connected component S is called *terminal* if for all $p \in S$ it holds that $p \xrightarrow{a} q$ implies $q \in S$. To improve the notation, we often label a transition of a Büchi automaton with an LTL() formula α meaning that there is a transition under each $e \in \Sigma$ satisfying α.

3.1 Almost linear Büchi automata (ALBA)

In this section we introduce a new kind of Büchi automata and describe its relation to the previously defined types.

Definition 2. 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*

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

such that $n \geq 0$, $\alpha_0, \alpha_1, \ldots, \alpha_n \in \mathrm{LTL}()$, *and for every* $q \in S$ *it holds that* $L(A_q) = L^{\Sigma}(\rho)$.

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

1. If we want to minimize the number of transitions and states of the automaton, we create just a "cycle" depicted on Figure 1.
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 to shorten the length of potential cycles in product automata, we add to the automaton A some shortcuts, see Figure 2.
3. If we want to minimize 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 3. Note that the number of states is exponential in the length of ρ, while it is only linear in the previous two cases.

In practice, the second kind of translation is usually chosen.

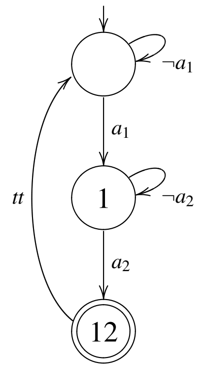

Figure 1: Minimal number of transitions.

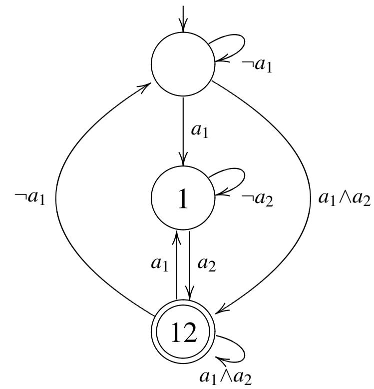

Figure 2: Minimal number of states and shortcuts.

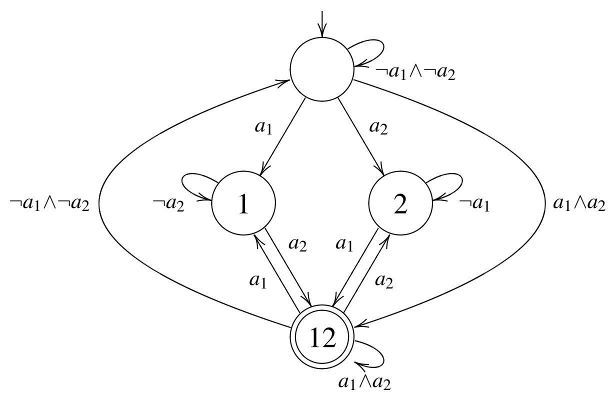

Figure 3: Shortest cycles in product automata.

3.2 Hierarchy of Büchi automata classes

Figure 4 depicts the hierarchy of the mentioned classes of Büchi automata. A line between two classes means that the upper class is strictly more expressible than the lower class. If the figure does not indicate such a relation between a pair of classes, then the classes are incomparable.

Indicated inclusions follow directly from definitions of the classes. The strictness of these inclusions is always easy to prove and the same holds also for the indicated incomparability relations. Note that only two of the considered classes can express the language of the formula $\mathsf{GF}a$: ALBA and the general class.

It is worth mentioning that the class of linear BA is expressively equivalent to negations of LTL^{det} formulae [7].

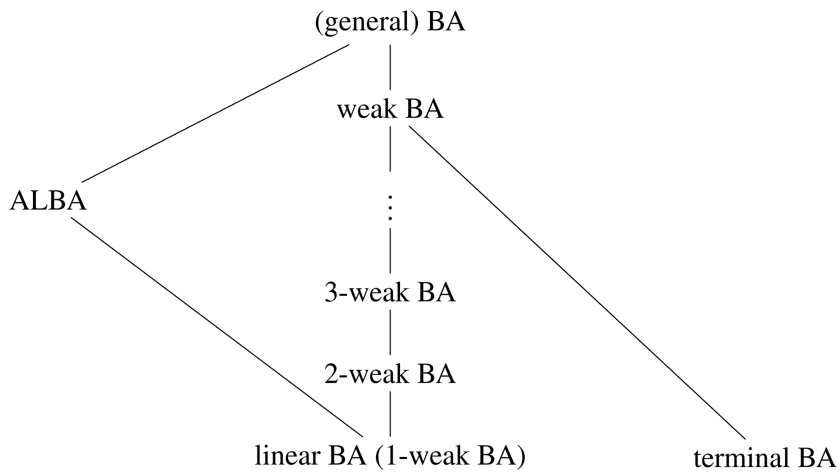

Figure 4: Hierarchy of Büchi automata classes.

4 Translation ALBA → LIO

Let $A = (\Sigma, Q, q_0, \delta, F)$ be an ALBA. For every state $q \in Q$, we recursively define a LIO formula $\varphi(q)$ such that $L(A_q) = L^{\Sigma}(\varphi(q))$. There are two cases:

- q is in a terminal strongly connected component. Due to the definition of ALBA, there exists a formula
$$\rho = \mathsf{G}\alpha_0 \wedge \bigwedge_{1 \le i \le n} \mathsf{GF}\alpha_i$$
such that $n \geq 0$, $\alpha_0, \alpha_1, \ldots, \alpha_n \in \mathrm{LTL}()$. We set $\varphi(q) = \rho$. Note that ρ is a formula of $\mathrm{LTL}(\mathsf{F}, \mathsf{G})$.

- 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 loops on q and $q \xrightarrow{b_1} q_1$, $q \xrightarrow{b_2} q_2$, ..., $q \xrightarrow{b_m} q_m$ be all transitions leading from q to other states. Then we set
$$\varphi(q) = \begin{cases} (\bigvee_{0<i\le n} a_i) \ \mathsf{U} \bigvee_{0<j\le m} (b_j \wedge \mathsf{X}\varphi(q_j)) & \text{if } q \notin F, \\ \Big((\bigvee_{0<i\le n} a_i) \ \mathsf{U} \bigvee_{0<j\le m} (b_j \wedge \mathsf{X}\varphi(q_j))\Big) \ \vee \ \mathsf{G} \bigvee_{0<i\le n} a_i & \text{if } q \in F. \end{cases}$$
Note that $\varphi(q)$ is in LIO assuming that all $\varphi(q_j)$ are in LIO.

The correctness of the recursion follows from the fact that A is linear (except the terminal components). The whole automaton then corresponds to the formula $\varphi(q_0)$.

5 Translation LIO → ALBA

In this section, we always assume that LIO formulae are in *positive form*, i.e. no temporal operator is in scope of any negation. Every LIO formula can be transformed into this form using the following equivalences.

$$\neg\mathsf{F}\varphi \equiv \mathsf{G}\neg\varphi \qquad \neg\mathsf{G}\varphi \equiv \mathsf{F}\neg\varphi \qquad \neg(\varphi_1 \wedge \varphi_2) \equiv \neg\varphi_1 \vee \neg\varphi_2 \qquad \neg(\varphi_1 \vee \varphi_2) \equiv \neg\varphi_1 \wedge \neg\varphi_2$$

For each LIO formula φ, we define its *size* as follows:

- if φ is in LTL(), we set $size(\varphi) = 1$,
- if φ is not in LTL(), we define its size recursively:

$$\begin{array}{rcl} size(\varphi_1 \vee \varphi_2) &=& size(\varphi_1) + 1 + size(\varphi_2) \\ size(\varphi_1 \wedge \varphi_2) &=& size(\varphi_1) + 1 + size(\varphi_2) \\ size(\mathsf{F}\varphi) &=& 1 + size(\varphi) \\ size(\mathsf{G}\varphi) &=& 2 * size(\varphi) \\ size(\mathsf{X}\varphi) &=& 1 + size(\varphi) \\ size(\alpha \,\mathsf{U}\, \varphi) &=& 1 + size(\varphi) \end{array}$$

Let S be a finite set of LIO formulae. We define its size as

$$\begin{array}{rcl} size(\emptyset) &=& (0, -) \\ size(S) &=& (k, (i_k, i_{k-1}, \dots, i_1)) \end{array}$$

where $k = max\{size(\varphi) \mid \varphi \in S\}$ and $i_j = |\{\varphi \mid \varphi \in S \wedge size(\varphi) = j\}|$ for each $k \geq j \geq 1$. Finally, we define a strict (lexicographical) order $<$ on sizes of these sets in the following way.

$$\begin{array}{rl} (k, (i_k, i_{k-1}, \dots, i_1)) < (l, (j_l, j_{l-1}, \dots, j_1)) \iff & k < l \;\vee \\ & (k = l \;\wedge\; \exists k \geq m \geq 1 \,.\, i_m < j_m \;\wedge\; \forall k \geq n > m \,.\, i_n = j_n) \end{array}$$

The translation is based on transformation of a LIO formula into an equivalent formula of a special form. Formally, to every LIO formula φ we assign a set $R(\varphi) \subseteq \text{LTL}() \times P_{fin}(\text{LIO})$, where $P_{fin}(\text{LIO})$ is the set of all finite subsets of LIO, such that

$$\varphi \;\equiv \bigvee_{(\alpha,S) \in R(\varphi)} (\alpha \;\wedge\; \mathsf{X} \bigwedge_{\sigma \in S} \sigma).$$

The set $R(\varphi)$ is defined recursively. The recursion is always bounded as each $R(\varphi')$ appearing in the definition of $R(\varphi)$ satisfies $size(\varphi') < size(\varphi)$. In the following, α always represents a formula of LTL(). We define $R(\varphi)$ according to the structure of φ.

- $\boxed{\alpha}$ $R(\alpha) = \{(\alpha, \emptyset)\}$
- $\boxed{\varphi_1 \vee \varphi_2}$ $R(\varphi_1 \vee \varphi_2) = R(\varphi_1) \cup R(\varphi_2)$
- $\boxed{\varphi_1 \wedge \varphi_2}$ $R(\varphi_1 \wedge \varphi_2) = \{(\alpha_1 \wedge \alpha_2, S_1 \cup S_2) \mid (\alpha_1, S_1) \in R(\varphi_1), (\alpha_2, S_2) \in R(\varphi_2)\}$
- $\boxed{\mathsf{F}\varphi_0}$ $R(\mathsf{F}\varphi_0) = \{(tt, \{\mathsf{F}\varphi_0\})\} \cup R(\varphi_0)$
- $\boxed{\mathsf{X}\varphi_0}$ $R(\mathsf{X}\varphi_0) = \{(tt, \{\varphi_0\})\}$
- $\boxed{\alpha \,\mathsf{U}\, \varphi_0}$ $R(\alpha \,\mathsf{U}\, \varphi_0) = \{(\alpha, \{\alpha \,\mathsf{U}\, \varphi_0\})\} \cup R(\varphi_0)$
- $\boxed{\mathsf{G}\varphi_0}$ This case is divided into the following subcases according to the structure of φ_0:
  - $\boxed{\alpha}$ $R(\mathsf{G}\alpha) = \{(\alpha, \{\mathsf{G}\alpha\})\}$
  - $\boxed{\varphi_1 \wedge \varphi_2}$ $R(\mathsf{G}(\varphi_1 \wedge \varphi_2)) = R(\mathsf{G}\varphi_1 \wedge \mathsf{G}\varphi_2)$

- $\boxed{\mathsf{F}\varphi_1}$ This case is again divided into the following subcases according to the structure of φ_1:
  * $\boxed{\alpha}$ $R(\mathsf{GF}\alpha) = \{(tt, \{\mathsf{GF}\alpha\})\}$
  * $\boxed{\varphi_3 \vee \varphi_4}$ $R(\mathsf{GF}(\varphi_3 \vee \varphi_4)) = R(\mathsf{GF}\varphi_3) \cup R(\mathsf{GF}\varphi_4)$
  * $\boxed{\varphi_3 \wedge \varphi_4}$ As conjunction is an associative operator, we can see it as an operator of arbitrary arity and we can assume that all conjuncts are not conjunctions. Then either all conjuncts are formulae of LTL() (i.e. $\varphi_3 \wedge \varphi_4 \in$ LTL() - this case has been already covered by the Case $\mathsf{GF}\alpha$), or at least one conjunct 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 conjunction of all the other conjuncts. We proceed according to the structure of φ_4.
    · $\boxed{\varphi_5 \vee \varphi_6}$ As $\mathsf{GF}(\varphi_3 \wedge (\varphi_5 \vee \varphi_6)) \equiv \mathsf{GF}(\varphi_3 \wedge \varphi_5) \vee \mathsf{GF}(\varphi_3 \wedge \varphi_6)$, we set $R(\mathsf{GF}(\varphi_3 \wedge (\varphi_5 \vee \varphi_6))) = R(\mathsf{GF}(\varphi_3 \wedge \varphi_5)) \cup R(\mathsf{GF}(\varphi_3 \wedge \varphi_6))$.
    · $\boxed{\mathsf{F}\varphi_5}$ As $\mathsf{GF}(\varphi_3 \wedge \mathsf{F}\varphi_5) \equiv (\mathsf{GF}\varphi_3) \wedge (\mathsf{GF}\varphi_5)$, we set $R(\mathsf{GF}(\varphi_3 \wedge \mathsf{F}\varphi_5)) = R((\mathsf{GF}\varphi_3) \wedge (\mathsf{GF}\varphi_5))$.
    · $\boxed{\mathsf{G}\varphi_5}$ As $\mathsf{GF}(\varphi_3 \wedge \mathsf{G}\varphi_5) \equiv (\mathsf{GF}\varphi_3) \wedge (\mathsf{GFG}\varphi_5) \equiv (\mathsf{GF}\varphi_3) \wedge (\mathsf{FG}\varphi_5)$, we set $R(\mathsf{GF}(\varphi_3 \wedge \mathsf{G}\varphi_5)) = R((\mathsf{GF}\varphi_3) \wedge (\mathsf{FG}\varphi_5))$.
  * $\boxed{\mathsf{F}\varphi_3}$ $R(\mathsf{GFF}\varphi_3) = R(\mathsf{GF}\varphi_3)$
  * $\boxed{\mathsf{G}\varphi_3}$ $R(\mathsf{GFG}\varphi_3) = R(\mathsf{FG}\varphi_3)$
- $\boxed{\varphi_1 \vee \varphi_2}$ The situation is similar to the Case $\mathsf{GF}(\varphi_2 \wedge \varphi_4)$. Hence, either $\varphi_1 \vee \varphi_2 \in$ LTL() (this has been already solved in Case $\mathsf{G}\alpha$), 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.
  * $\boxed{\varphi_3 \wedge \varphi_4}$ As $\mathsf{G}(\varphi_1 \vee (\varphi_3 \wedge \varphi_4)) \equiv \mathsf{G}(\varphi_1 \vee \varphi_3) \wedge \mathsf{G}(\varphi_1 \vee \varphi_4)$, we set $R(\mathsf{G}(\varphi_1 \vee (\varphi_3 \wedge \varphi_4))) = \{(\alpha_1 \wedge \alpha_2, S_1 \cup S_2) \mid (\alpha_1, S_1) \in R(\mathsf{G}(\varphi_1 \vee \varphi_3)), (\alpha_2, S_2) \in R(\mathsf{G}(\varphi_1 \vee \varphi_4))\}$.
  * $\boxed{\mathsf{F}\varphi_3}$ As $\mathsf{G}(\varphi_1 \vee \mathsf{F}\varphi_3) \equiv (\mathsf{G}\varphi_1) \vee \mathsf{F}(\varphi_3 \wedge \mathsf{XG}\varphi_1) \vee \mathsf{GF}\varphi_3 \equiv (\mathsf{G}\varphi_1) \vee tt\,\mathsf{U}\,(\varphi_3 \wedge (\mathsf{XG}\varphi_1)) \vee \mathsf{GF}\varphi_3$, we set $R(\mathsf{G}(\varphi_1 \vee \mathsf{F}\varphi_3)) = R(\mathsf{G}\varphi_1) \cup R(tt\,\mathsf{U}\,(\varphi_3 \wedge (\mathsf{XG}\varphi_1))) \cup R(\mathsf{GF}\varphi_3)$.
  * $\boxed{\mathsf{G}\varphi_3}$ $R(\mathsf{G}(\varphi_1 \vee \mathsf{G}\varphi_3))$: Here we consider only the following two structures of the whole subformula $\varphi_1 \vee \mathsf{G}\varphi_3$ (the other possibilities fit to some of the previous cases):
    · $\boxed{\bigvee_{\varphi' \in G} \mathsf{G}\varphi'}$ As $\mathsf{G}(\bigvee_{\varphi' \in G} \mathsf{G}\varphi') \equiv \bigvee_{\varphi' \in G}(\mathsf{G}\varphi')$, we set $R(\mathsf{G}(\bigvee_{\varphi' \in G} \mathsf{G}\varphi')) = \bigcup_{\varphi' \in G} R(\mathsf{G}\varphi')$.
    · $\boxed{\alpha \vee \bigvee_{\varphi' \in G} \mathsf{G}\varphi'}$ As $\mathsf{G}(\alpha \vee \bigvee_{\varphi' \in G} \mathsf{G}\varphi') \equiv$ $(\mathsf{G}\alpha) \vee \bigvee_{\varphi' \in G}(\mathsf{G}\varphi') \vee \bigvee_{\varphi' \in G}(\alpha \wedge \mathsf{XG}(\alpha \vee \mathsf{G}\varphi')$, we set $R(\mathsf{G}(\alpha \vee \bigvee_{\varphi' \in G} \mathsf{G}\varphi')) = R(\mathsf{G}\alpha) \cup \bigcup_{\varphi' \in G} R(\mathsf{G}\varphi') \cup \bigcup_{\varphi' \in G}\{(\alpha, \{\mathsf{G}(\alpha \vee \mathsf{G}\varphi')\})\}$.
- $\boxed{\mathsf{G}\varphi_1}$ $R(\mathsf{GG}\varphi_1) = R(\mathsf{G}\varphi_1)$

Moreover, for every finite set of LIO formulae, we define

$$R(S) = R(\bigwedge_{\varphi \in S} \varphi).$$

In particular, $R(\emptyset) = \{(tt, \emptyset)\}$.

Before we provide the construction of ALBA automaton for a given LIO formula, we mention some crucial observations. First of all, one can readily confirm the following observation.

Lemma 3. *For every $(\alpha, S) \in R(\varphi)$ it holds that, for each $\varphi' \in S$, either $\varphi' = \varphi$ or $size(\varphi') < size(\varphi)$.*

In fact, there are only five cases where $\varphi' = \varphi$, namely if φ has the form $\mathsf{F}\varphi_0$ or $\alpha\,\mathsf{U}\,\varphi_0$ or $\mathsf{G}\alpha$ or $\mathsf{GF}\alpha$ or $\mathsf{G}(\alpha \vee \mathsf{G}\varphi')$ (this is a special case of the form $\mathsf{G}(\alpha \vee \bigvee_{\varphi' \in G} \mathsf{G}\varphi')$). Lemma 3 together with an analysis of the listed cases directly implies the following property.

Lemma 4. *For every $(\alpha, S) \in R(\{\varphi\})$, either $S = \{\varphi\}$ or $size(S) < size(\{\varphi\})$.*

This lemma immediately implies the following one.

Lemma 5. *Let S be a finite set of LIO formulae in positive form. For every $(\alpha, S') \in R(S)$ it holds that $S = S'$ or $size(S') < size(S)$.*

If we look at the five cases mentioned above Lemma 4, we can easily see that only two of them have the property that $R(\varphi) = (\alpha, \varphi)$, namely the cases $\mathsf{G}\alpha$ and $\mathsf{GF}\alpha$. This is the crucial argument for the following observation.

Lemma 6. *Let S be a finite set of LIO formulae in positive form. It holds that*

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

Now we are ready to finish the translation. Let φ be a LIO formula in positive normal form and let AP' be the set of all atomic propositions occurring in φ. We describe the ALBA automaton in a concise form: terminal components will be described by distinguished states labelled with the corresponding LTL formulae of the form $\rho = \mathsf{G}\alpha_0 \wedge \bigwedge_{1 \le i \le n} \mathsf{GF}\alpha_i$. A standard ALBA can be obtained from this concise form very easily: we just replace every such a state by a corresponding component (as indicated in Figures 1, 2, or 3).

The automaton corresponding to φ is constructed as $(\Sigma, Q, q_0, \delta, F)$, where

- $\Sigma = 2^{AP'}$,
- $Q = 2^M$ and $M = \{\varphi' \mid \varphi'$ is a LIO formula over AP' and $size(\varphi') \le size(\varphi)\}$ is a set subsuming all formulae that can be derived from φ by repeated applications of $R(\cdot)$ (see Lemma 3),
- $q_0 = \{\varphi\}$,
- For each $e \in \Sigma$ and $S \in Q$, we set $\delta(S, e) = \{S' \mid (\alpha, S') \in R(S)$ and $e \models \alpha\}$,
- accepting states appear only in terminal components. Due to Lemma 6, terminal components correspond to states S satisfying $S \subseteq \{\mathsf{G}\alpha, \mathsf{GF}\alpha \mid \alpha \in \mathrm{LTL}()\}$. Hence, we label such a state S with the formula
$$(\mathsf{G} \bigwedge_{\mathsf{G}\alpha \in S} \alpha) \wedge \bigwedge_{\mathsf{GF}\alpha \in S} \mathsf{GF}\alpha$$
of the desired form.

The language equivalence between φ and the constructed automaton follows from the properties of $R(\cdot)$. The constructed automaton is ALBA due to Lemma 5 (linearity except terminal components) and Lemma 6 (condition on terminal components). Note that the translation directly provides triple exponential bound on the size of Q in the length of φ (even $size(\varphi)$ can be exponential in the length of φ). However, we conjecture that the size of Q is in fact only singly exponential in the length of φ.

A natural question is whether standard translations of LTL into BA also produce ALBA when applied to LIO. The answer is negative. For example, Gastin and Oddoux's popular implementation of the translation via alternating automata [4] (available online at `http://www.lsv.ens-cachan.fr/~gastin/ltl2ba/index.php`) transforms the LIO formula $\mathsf{G}(\mathsf{G}(a \vee \mathsf{F}b) \vee \mathsf{G}(c \vee \mathsf{F}d))$ into a Büchi automaton that is not ALBA (it contains nonterminal strongly connected components of size greater than one).

6 Conclusion

We have introduced a new class of Büchi automata called *almost linear Büchi automata (ALBA)*. We have compared the expressive power of ALBA with other classes of Büchi automata. Further, we have identified a fragment of LTL called *LIO* and equivalent to ALBA. The LIO fragment subsumes some previously studied LTL fragments, in particular the *restricted temporal logic* and negations of LTL^{det}, i.e. the common fragment of CTL and LTL. We have provided a direct translation of LIO formulae into Büchi automata (BA). In contrast to standard translations of LTL into BA, our translation does not use any intermediate formalism and always produces ALBA. We expect that the specific structure of ALBA can lead to development of algorithms designed especially for model checking of negations of LIO properties. To emphasize potential usability of such algorithms, we have analysed the collection of the most often verified properties called Specification Patterns [3]. It shows up that negations of 89% of the properties can be expressed as LIO formulae and hence translated to ALBA.

References

[1] Ivana Černá & Radek Pelánek (2003): *Relating Hierarchy of Temporal Properties to Model Checking*. In: *Proceedings of the 30th Symposium on Mathematical Foundations of Computer Science (MFCS'03)*, *Lecture Notes in Computer Science* 2747. Springer-Verlag, pp. 318–327.

[2] Dennis R. Dams (1999): *Flat Fragments of CTL and CTL*: Separating the Expressive and Distinguishing Powers*. *Logic Journal of the IGPL* 7(1), pp. 55–78.

[3] Matthew B. Dwyer, George S. Avrunin & James C. Corbett (1998): *Property Specification Patterns for Finite-State Verification*. In: *Proc. 2nd Workshop on Formal Methods in Software Practice (FMSP-98)*. ACM Press, New York, pp. 7–15.

[4] Paul Gastin & Denis Oddoux (2001): *Fast LTL to Büchi Automata Translation*. In: G. Berry, H. Comon & A. Finkel, editors: *Proceedings of the 13th International Conference on Computer Aided Verification (CAV'01)*, *Lecture Notes in Computer Science* 2102. Springer-Verlag, pp. 53–65.

[5] Rob Gerth, Doron Peled, Moshe Y. Vardi & Pierre Wolper (1995): *Simple On-the-fly Automatic Verification of Linear Temporal Logic*. In: *Protocol Specification Testing and Verification*. Chapman & Hall, pp. 3–18.

[6] Leslie Lamport (1983): *What good is Temporal Logic?* In: R. E. A. Mason, editor: *Proceedings of the IFIP Congress on Information Processing*. North-Holland, Amsterdam, pp. 657–667.

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

[8] Zohar Manna & Amir Pnueli (1990): *A hierarchy of temporal properties*. In: *Proceedings of ACM Symposium on Principles of Distributed Computing (PODC'90)*. ACM Press, pp. 377–410.

[9] David E. Muller, Ahmed Saoudi & Paul E. Schupp (1988): *Weak alternating automata give a simple explanation of why most temporal and dynamic logics are decidable in exponential time*. In: *Proceedings of the 3rd Annual IEEE Symposium on Logic in Computer Science (LICS'88)*. IEEE Computer Society Press, pp. 422–427.

[10] Dominique Perrin & Jean-Eric Pin (2004): *Infinite words*, *Pure and Applied Mathematics* 141. Elsevier.

[11] Amir Pnueli (1977): *The temporal logic of programs*. In: *Proc. 18th IEEE Symposium on the Foundations of Computer Science*. pp. 46–57.

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

[13] Moshe Y. Vardi (1995): *An Automata-Theoretic Approach to Linear Temporal Logic*. In: *Banff Higher Order Workshop*, *Lecture Notes in Computer Science* 1043. Springer, pp. 238–266.