From a Subset of LTL Formula to Büchi Automata

Bilal Kanso

Lebanese University
Faculty of sciences (V), Computer Science Department
Email: bilal_kanso@hotmail.com

Ali Kansou

Lebanese University
Faculty of sciences (V), Computer Science Department
Email: ali.kansou@gmail.com

***Abstract*—We present a fragment of Linear Temporal Logic (LTL) together with an polynomial translation of formula from this LTL fragment into equivalent Büchi automata. The translation is completely implemented based on Java Pluging Framework in GOAL Tool as a plugin. The implementation is mainly based on pre-proven theorems such that the transformation works very efficiently. In particular, it runs in polynomial space in terms of the length of the given formula. The main application of this transformation could be in model checking area which consists in obtaining a Büchi automaton that is equivalent to the software system specification and another one that is equivalent to the negation of the property. The intersection of the two Büchi automata is empty if the model satisfies the property. Furthermore, the experiments are performed with three sets of LTL formula, which is commonly used in the literature and the result shows that our proposed LTL fragment covers most of them.**

***Keywords*–Linear Temporal Logic; Büchi automata; Model checking; Compositional modeling.**

I. INTRODUCTION

The Linear Temporal Logic (LTL) [1] becomes increasingly one of the most important formalisms to model system properties which are widely used in different areas such as model checking [2][3], testing [4][5], reasoning event in time, *etc.* It is equivalent to first-order logic over finite and infinite words. It is well-known that model checking and satisfiability for LTL are PSPACE-complete and in most all cases the model checking problem is equivalent to a satisfiability-checking problem. This justifies why the satisfiability problem for LTL and its fragments has received so much attention. By way of illustration, model checking based on LTL formalism is PSPACE-hard [6][7]. This complexity arises from the translation step of the negation of a property (described as a LTL formulæ) into Büchi automata. Indeed, the Büchi automaton of a property is constructed in exponential space in the length of this property. This makes verification methods hard or even impossible to be implemented in practice and makes the scalability of the LTL model checking limited, which commonly referred to as the state explosion problem [8].

The question we handled is there some LTL fragments that are feasible in practice. In this paper, we contribute to finding a subset of LTL properties that can be converted polynomially into Büchi automata. A fragment called, *FLTL Logic*, is defined and how formula in this fragment can be transformed into Büchi automata whose the state space size is linear is shown. This fragment is identified by looking for natural subclasses of LTL formula for which complexity decreases and by deep understanding of what makes the converting into Büchi automata PSPACE-complete. Thanks to the structure of our fragment *FLTL* formulæ, the proposed algorithm can be compositional in the sense that the target Büchi automaton associated to a given formulæ is obtained by developing a sub-automaton for each sub-formulæ of the principal formulæ. Hence, the basic idea for developing the final automaton for a FLTL formulæ φ is that φ can be recursively decomposed into a set of sub-formula, arriving at sub-formula that can be completely handled. Composition is then used for assembling different sub-automaton and then forming larger ones. Such a composition can be seen as an operation taking sub-automata for sub-formula, as well as the FLTL operator to provide a new more complex automaton. Furthermore, we showed by experiments that the fragment coverage average is 65.531% which is acceptable and slightly high and the use of such fragments seems promising. The experiments are based on three common sets of LTL formula widely used in the literature. For each set, we identify the formula which can be described in the extension and generate its equivalent automata using the proposed algorithm.

The rest of this article is organized as follows: Section II briefly describes Büchi automata. In Section III, we describe our fragment of LTL logic and the reasons to choose it. In Section IV, we present for each formulæ in our fragment LTL, its equivalent Büchi automata. Section V shows the final algorithm that generates to any formulæ in our fragment an equivalent Büchi automaton. Section VI represents the experiments we conducted to compute the coverage average of our LTL fragment. Section VII presents the related work and Section VIII presents the conclusion and some future works.

II. BÜCHI AUTOMATA

A Büchi automaton is variant of non-deterministic finite-state automata on infinite inputs [9]-[10]. A word is accepted if the automaton goes through some designated "accept" states infinitely often while reading it. Formally, a **Büchi automaton** is defined by a 5-tuple $A = (S, s_0, F, \Sigma, \delta)$ where S is a finite set of states, $s_0 \in S$ is the initial state, Σ is a non-empty set of atomic propositions, $F \subseteq S$ is a finite set of accepting states and $\delta : S \times \Sigma \longrightarrow 2^S$ is a transition function. A **run** of A on $\sigma = \sigma(0)\sigma(1)\sigma(2)\cdots \in \Sigma^\omega$ is an infinite sequence of states $s_0 s_1 s_2 \cdots \in S^\omega$ starting with the initial state s_0 of A such that $\forall i, i \geq 0, s_{i+1} \in \delta(s_i, \sigma(i))$. A run $s_0 s_1 s_2 \ldots$ is **accepting** by an automaton A if A goes through accepting states (i.e $\in F$) infinitely often while reading it. The *accepted language* of a Büchi automaton A, denoted by, $\mathcal{L}_\omega(A)$ is then defined by $\mathcal{L}_\omega(A) = \{\sigma \in \Sigma^\omega \mid \text{there is an accepting run for } \sigma \text{ in } A\}$. The union of two Büchi automata A_1 and A_2 is formally defined as follows:

Definition 1 (Buchi automata union): Let $A_1 = (S_1, s_{10}, F_1, \Sigma, \delta_1)$ and $A_2 = (S_2, s_{20}, F_2, \Sigma, \delta_2)$ be two Büchi automata. The **union** $A_1 \cup A_2$ of A_1 and A_2 is the Büchi automaton $A = (S, s_0, F, \Sigma, \delta)$ defined as follows:

- $S = S_1 \cup S_2 \cup \{s_0\}$
- $s_0 \in S$ is the initial state
- $F = F_1 \cup F_2$
- the transition relation δ is defined as follows:

$$\delta(s,p) = \begin{cases} \delta_1(s,p) \text{ if } s \in S_1 \\ \delta_2(s,p) \text{ if } s \in S_2 \\ \delta_1(s_{10},p) \cup \delta_2(s_{20},p) \text{ if } s \text{ is the initial state } s_0 \end{cases}$$

In Definition 1, we add a new initial (nonaccept) state s_{new} to the union set of states of both A_1 and $A_2$and the transitions $s_{\mathsf{new}} \xrightarrow{p} s$ if and only if $s^0_{A_1} \xrightarrow{p} s$ and $s_{\mathsf{new}} \xrightarrow{p} s$ if and only if $s^0_{A_2} \xrightarrow{p} s$ to the union set of transitions of both A_1 and A_2.

The construction of the intersection automaton works a little differently from the finite state automata case. One needs to check whether both sets of accepting states are visited infinitely often. Consider two runs r_1 and r_2 and a word σ where r_1 goes through an accept state after $\sigma(0), \sigma(2), \ldots$ and r_2 enters accept state after $\sigma(0)\sigma(3)\ldots$. Thus, there is no guarantee that r_1 and r_2 will enter accept states simultaneously. To overcome this problem, we need to identify the accept states of the intersection of the two automata. To do so, we create two copies of the intersected state space. In the first copy, we check for occurrence of the first acceptance set. In the second copy, we check for occurrence of the second acceptance set. When a run enters a final state in the first copy, we wait for that run also enters in an accept state in the second copy. When this is encountered, we switch back to the first copy and so on. We repeat jumping back and forth between the two copies whenever we find an accepting state.

Definition 2 (Buchi automata intersection): Let $A_1 = (S_1, s_{10}, F_1, \Sigma, \delta_1)$ and $A_2 = (S_2, s_{20}, F_2, \Sigma, \delta_2)$ be two Büchi automata. The intersection $A_1 \cap A_2$ of A_1 and A_2 is the Büchi automaton $A = (S, s_0, F, \Sigma, \delta)$ defined as follows:

- $S = S_1 \times S_2 \times \{1, 2\}$
- $s_0 = (s_{10}, s_{20}, 1)$
- $F = S_1 \times F_2 \times \{2\}$
- The transition function δ is defined as follows:

$$\delta((s_1, s_1', 1), p) = \begin{cases} (s_2, s_2', 1) \text{ if } s_2 \in \delta_1(s_1, p), \\ s_2' \in \delta_2(s_2, p) \text{ and } s_1 \notin F_1 \\ (s_2, s_2', 2) \text{ if } s_2 \in \delta_1(s_1, p), \\ s_2' \in \delta_2(s_2, p) \text{ and } s_1 \in F_1 \end{cases}$$

$$\delta((s_1, s_1', 2), p) = \begin{cases} (s_2, s_2', 2) \text{ if } s_2 \in \delta_1(s_1, p), \\ s_2' \in \delta_2(s_2, p) \text{ and } s_1' \notin F_2 \\ (s_2, s_2', 1) \text{ if } s_2 \in \delta_1(s_1, p), \\ s_2' \in \delta_2(s_2, p) \text{ and } s_1' \in F_2 \end{cases}$$

Theorem 1: Let $\psi = \varphi_1 \vee \varphi_2$ (resp. $\psi = \varphi_1 \wedge \varphi_2$) be a LTL formulæ and A_{φ_i} be the Büchi automaton equivalent to φ_i for $i = 1, 2$. Let A_ψ be the LTL automaton built according to Definition 1 (resp. Definition 2). Then, $\mathsf{Words}(\psi) = \mathcal{L}_\omega(A_\psi)$ (See Proof in Appendix)

III. Flat LTL Logic

In this section, we introduce our subset of LTL logic that we call *FLTL Logic*. This fragment will be used to express temporal properties and then translate them into Büchi automata in linear size. The syntax of our FLTL logic adds to usual boolean propositional operators $\neg$ (negation) and $\wedge$ (conjunction), some modal operators that describe how the behavior changes with time. **Next**: $\mathsf{X}\varphi$ requires that the formula φ be true in the next state. **Until**: $\varphi_1 \mathsf{U} \varphi_2$ requires that the formula φ_1 be true *until* the formula φ_2 is true, which is required to happen. **Eventually**: $\Diamond\varphi$ requires that the formula φ be true at some point in the future (starting from the present) and it is equivalent to $\Diamond\varphi \equiv \mathsf{true} \mathsf{U} \varphi$. **Always**: $\Box\varphi$ requires that the formula φ be true at every point in the future (including the present). **Release**: $\varphi_1 \mathsf{R} \varphi_2$ requires that its second argument φ_2 always be true, a requirement that is *released* as soon as its first argument φ_1 becomes true. It is equivalent to $\varphi_1 \mathsf{R} \varphi_2 \equiv \neg(\neg\varphi_1 \mathsf{U} \neg\varphi_2)$.

A. Our fragment LTL logic

Definition 3 (FLTL formulæ): **The set of FLTL formulæ** $\mathcal{L}_f$ is given by the following grammar:

$$\varphi{:=}\Theta \mid \Box\Theta \mid \Theta \mathsf{U} \varphi \mid \varphi \mathsf{R} \Theta \mid \mathsf{X}\varphi \mid \neg\Delta \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2$$

where Θ is a propositional formula defined by: $\Theta{:=}\mathsf{true} \mid p \mid \neg\Theta \mid \Theta_1 \wedge \Theta_2$ and Δ is the formula defined by: $\Delta{:=}\Delta \mathsf{U} \Theta \mid \Theta \mathsf{R} \Delta \mid \mathsf{X}\varphi \mid \neg\Delta$ with $p \in \Sigma$.

For the sake of brevity and the lack of space, we only discuss here why the fragment $\Theta \mathsf{U} \varphi$ is included within our LTL fragment to the detriment of both formula $\varphi_1 \mathsf{U} \varphi_2$ and $\varphi_1 \mathsf{U} \Theta$. It is well-known the size of an Büchi automaton $\overline{A}$ that recognizes the complement language $\mathcal{L}_\omega(\overline{A})$ of the language accepted $\mathcal{L}_\omega(A)$ by an automaton A is exponential [11], [12]. Suppose we have separately built an automaton A_1 for φ_1 and an automaton A_2 for φ_2, and let us then try to compositionally obtain the resulting automaton A for φ. According to the until operator's semantics, it is required that φ *holds at the current moment, if there is some future moment for which* φ_2 *holds and* φ_1 *holds at all moments until that future moment*. That means constructing the automaton for $\varphi = \varphi_1 \mathsf{U} \varphi_2$ firstly requires constructing of the intersection of A_1 and $\overline{A_2}$. As stated previously, computing $\overline{A_2}$ is exponential and therefore, constructing the Büchi automaton for $\varphi \mathsf{U} \varphi_2$ is exponential. To avoid this kind of formula, we choose the formulæ $\Theta \mathsf{U} \varphi$ to be a part of our LTL subset where the construction of the Büchi automaton associated to it, does not need to complement any Büchi automaton.

B. Positive Normal Form (FPNF)

As LTL formula, FLTL formula can be transformed into the so-called *Positive Normal form (FPNF)*. This form is characterized by the fact that negations only occur adjacent to atomic propositions. All negation symbols of the given LTL formula have to be pushed inwards over the temporal operators.

Definition 4 (FPNF): **The set of FLTL Positive Normal Form (FPNF) formulæ** $\mathcal{L}_{FPNF}$ is given by the following grammar:

$$\varphi{:=}\mathsf{true} \mid p \mid \neg p \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \Box\Theta \mid \Theta \mathsf{U} \varphi \mid \varphi \mathsf{R} \Theta \mid \mathsf{X}\varphi$$

Each formulæ $\varphi \in \mathcal{L}_f$, can be transformed into a formulæ $\varphi\prime \in \mathcal{L}_{FPNF}$. This is done by pushing negations inside, near to atomic propositions. To do this, we use the following transformation rules:

$\neg\mathsf{true} \rightsquigarrow \mathsf{false}, \neg\neg\varphi \rightsquigarrow \varphi, \neg(\varphi_1 \wedge \varphi_2) \rightsquigarrow \neg\varphi_1 \vee \neg\varphi_2, \neg\mathsf{X}\varphi \rightsquigarrow \mathsf{X}\neg\varphi, \neg(\varphi \mathsf{~U~} \Theta) \rightsquigarrow \neg\varphi \mathsf{~R~} \neg\Theta, \neg(\Theta \mathsf{~R~} \varphi) \rightsquigarrow \neg\Theta \mathsf{~U~} \neg\varphi$.

Theorem 2: For any FLTL formulæ $\varphi \in \mathcal{L}_f$, there exists an equivalent LTL formula $\varphi\prime \in \mathcal{L}_{FPNF}$ $|\varphi\prime| = \mathcal{O}(|\varphi|)$.

C. Semantics

The semantics of FLTL formulæ is defined over infinite sequences $\sigma : \mathbb{N} \longrightarrow 2^{\Sigma}$ (2^{Σ} is the power set of Σ). In other words, a model is an infinite sequence $A_0 A_1 \ldots$ of subsets of Σ. The function σ, called *interpretation function*, describes how the truth of atomic propositions changes as time progresses. For every sequence σ, we write $\sigma = (\sigma(0), \ldots, \sigma(n), \ldots)$. Thus, $\sigma(\mathbf{i})$ denotes the state at index i and $\sigma(i : j)$ the part of σ containing the sequence of states between i and j. $\sigma(\mathbf{i}\ldots) = \mathbf{A_i A_{i+1} A_{i+2}} \ldots$ denotes the suffix of a sequence $\sigma = A_0 A_1 A_2 \cdots \in (2^{\Sigma})^{\omega}$ starting in the $(i+1)$st symbol A_i where ω denotes *infinity*. We also write $\sigma(i) \models \varphi$ to denote that "*φ is true at time instant i in the model σ*". This notion is defined inductively, according to the structure of φ.

The FLTL formula are interpreted over infinite sequences of states $\sigma : \mathbb{N} \longrightarrow 2^{\Sigma}$ as follows:

Definition 5 (Semantics of FLTL): Let $\sigma : \mathbb{N} \longrightarrow 2^{\Sigma}$ be an interpretation function and $\varphi \in \mathcal{L}_{FLTL}$. σ **satisfies** φ, noted $\sigma \models \varphi$,is inductively defined over the construction of φ as follows:

- $\varphi = \mathsf{true}$, then $\sigma \models \mathsf{true}$
- if $\varphi = p$, then $\sigma \models p$ iff $p \in \sigma(0)$
- if $\varphi = \mathsf{X}\varphi'$, then $\sigma \models \mathsf{X}\varphi'$ iff $\sigma(1) \models \varphi'$
- if $\varphi = \Box\Theta$, then $\sigma \models \Box\Theta$ iff $\forall i \geq 0, \sigma(i) \models \Theta$
- if $\varphi = \Theta \mathsf{~U~} \varphi$, then $\sigma \models \Theta \mathsf{~U~} \varphi$ iff $\exists i, i \geq 0, \sigma(i, \ldots) \models \varphi$ and $\forall j, 0 \leq j < i, \sigma(j\ldots) \models \Theta$
- if $\varphi = \varphi \mathsf{~R~} \Theta$, then $\sigma \models \varphi \mathsf{~R~} \Theta$ iff $\exists i, i \geq 0, \sigma(i, \ldots) \models \varphi$ and $\forall j, j \geq 0, \sigma(j\ldots) \models \Theta$ or $\exists i, i \geq 0$ $(\sigma(i\ldots) \models \varphi \wedge \forall k, k \leq i, \sigma(k\ldots) \models \Theta)$
- if $\varphi = \neg\varphi'$, then $\sigma \models \neg\varphi'$ iff $\sigma \not\models \varphi'$
- Propositional connectives are handled as usual

The semantics of a FLTL formulæ can be also seen as the language $\mathsf{Words}(\varphi)$ that contains all infinite words over the set of atomic propositions (*i.e.* alphabet) 2^{Σ} that satisfy φ. Thus, the language $\mathsf{Words}(\varphi)$ for a FLTL formulæ φ is formally defined by $\mathsf{Words}(\varphi) = \{\sigma \in (2^{\Sigma})^{\omega} \mid \sigma \models \varphi\}$.

Proposition 1: Two FLTL formula φ_1 and φ_2 are *equivalent*, denoted $\varphi_1 \equiv \varphi_2$, if $\mathsf{Words}(\varphi_1)$ =$\mathsf{Words}(\varphi_2)$.

IV. Construction of Büchi automata for FLTL logic

In the sequel, we explain for each subformulæ in our fragment LTL logic how its equivalent Büchi automaton can be obtained.

A. Büchi automata for Θ formula

The Büchi automaton associated to a propositional formulæ Θ is obtained by creating two states s_0 and s_1 and two transitions tr_1 and tr_2. s_0 is the only initial state while s_1 is the only final state. tr_1 is the transition from s_0 to s_1 labeling with Θ while the transition tr_2 is a loop labeled with true over the state s_2.

Definition 6 (Θ automaton): Let Θ be a propositional formulæ. The **automaton** $A_{\Theta} = (S_{\Theta}, s^0_{\Theta}, F_{\Theta}, \Sigma, \delta_{\Theta})$ associated to Θ is defined as follows:

- $S_{\Theta} = \{s_0, s_1\}$, $s^0_{\Theta} = s_0$, $F_{\Theta} = \{s_1\}$
- The transition function δ is defined as follows:

$$\delta_{\Theta}(s_0, \Theta) = \{s_1\} \text{ and } \delta_{\Theta}(s_1, \mathsf{true}) = \{s_1\}$$

B. Büchi automata for Θ U φ formula

The automaton associated to $\Theta \mathsf{~U~} \varphi$ is obtained by adding a new initial (nonaccept) state s_{new} to the state set of A_{φ}, a loop over the added state s_{new} labeled with the propositional formula Θ and transitions $s_{\mathsf{new}} \xrightarrow{p} s$ if and only if and only if $s^0 \xrightarrow{p} s$ with s^0 is the initial state of A_{φ}. All other transitions of A_{φ}, as well as the accept states, remain unchanged. s_{new} is the single initial state automaton, is not accept, and has no incoming transitions except the loop one.

Definition 7 (Θ U φ automaton): Let Θ be a propositional formula and φ be an LTL flat formulæ. Let $A_{\varphi} = (S_{\varphi}, s^0_{\varphi}, F_{\varphi}, \Sigma, \delta_{\varphi})$ be the automaton associated to φ. The **automaton** $A_{\psi} = (S_{\psi}, s^0_{\psi}, F_{\psi}, \Sigma, \delta_{\psi})$ associated to $\psi = \Theta \mathsf{~U~} \varphi$ is defined as follows:

- $S_{\psi} = \{s_{\mathsf{new}}\} \cup S_{\varphi}$
- $s^0_{\psi} = s_{\mathsf{new}}$, $F_{\psi} = F_{\varphi}$
- The transition function δ_{ψ} is defined as follows:

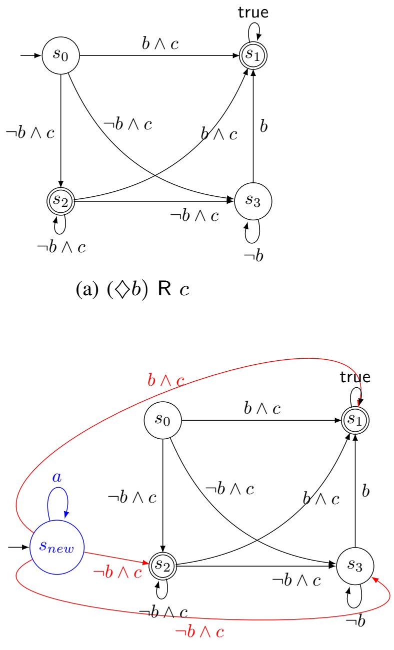
$$\delta_{\psi}(s,p) = \begin{cases} \delta_{\varphi}(s,p) \text{ if } s \in S_{\varphi} \ (A_{\varphi} \text{ transitions}) \\ \delta_{\varphi}(s^0_{\varphi}, p) \text{ if } s = s_{\mathsf{new}} \\ \quad \text{(Connection initial state to } A_{\varphi}) \\ \{s_{\mathsf{new}}\} \text{ if } s = s_{\mathsf{new}} \text{ and } p = \Theta \\ \quad \text{(Loop over the new initial state)} \end{cases}$$

Example 1: Figure 1 illustrates the composition definition of $\Theta \mathsf{~U~} \varphi$. Figure 1a shows the Büchi automaton associated to $(\Diamond b) \mathsf{~R~} c$. To construct the Büchi automaton associated to $(a \mathsf{~U~} (\Diamond b \mathsf{~R~} c))$, we add a new state s_{new} that we consider as initial state. Then, for each transition outgoing from s_{new} with label l and goes to state s, we add a transition from s_{new} to the state s with a label l. Finally, we then add a loop labeled with the atomic proposition a over the added state.

Theorem 3: Let $\psi = \Theta \mathsf{~U~} \varphi$, A_{φ} be the Büchi automaton equivalent to φ and A_{ψ} be the automaton built according to Definition 7. Then, $\mathsf{Words}(\psi) = \mathcal{L}_{\omega}(A_{\psi})$.

C. Büchi automata for Xφ formula

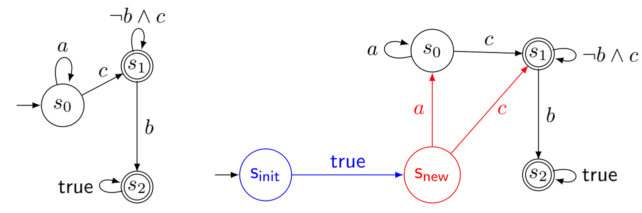
The automaton associated to $\mathsf{X}\varphi$ is obtained by adding two new states s_{new} (neither initial state or accept state) and s_{init} (considered as the initial state) to the state set of A_{φ} with the following two transitions (1) add for any transition in A_{φ} which starts from the initial state s^0 to a state s, a transition from s_{new} to s; (2) add a transition from the initial state s_{init} to the s_{new} labeled with true. All other transitions of A_{φ} remain



(a) $(\Diamond b)$ R c

(b) a U $(\Diamond b$ R $c)$

Figure 1. Example of composition: Θ U φ



(a) a U (Xb R c)

(b) X(a U (Xb R c))

Figure 2. Example of composition: Xφ formula

unchanged and final states of A_φ become accept ones of A_ψ and initial state of A_ψ become the state s_{init}.

Definition 8 (Xφ automaton): Let φ be an Flat LTL formulæ. Let $A_\varphi = (S_\varphi, s^0_\varphi, F_\varphi, \Sigma, \delta_\varphi)$ be the automaton equivalent to φ. The **automaton** $A_\psi = (S_\psi, s^0_\psi, F_\psi, \Sigma, \delta_\psi)$ equivalent to $\psi = \mathsf{X}\varphi$ is defined as follows:

- $S_\psi = S_\varphi \cup \{s_{\mathsf{new}}, s_{\mathsf{init}}\}$
- $s^0_\psi = s_{\mathsf{init}}$, $F_\psi = F_\varphi$
- The transition function δ is defined as follows:

$$\delta_\psi(s,p) = \begin{cases} \delta_\varphi(s,p) \text{ if } s \in S_\varphi \text{ } (A_\varphi \text{ transitions}) \\ \delta_\varphi(s^0_\varphi, p) \text{ if } s = s_{\mathsf{new}} \\ (\text{Connection } s_{\mathsf{new}} \text{ to initial state of } A_\varphi) \\ \{s_{\mathsf{new}}\} \text{ if } s = s_{\mathsf{init}} \text{ and } p = \mathsf{true} \\ (\text{Connection } s_{\mathsf{init}} \text{ to } s_{\mathsf{new}}) \end{cases}$$

Example 2: Figure 2 illustrates the definition of Xφ. Figure 2a shows the Büchi automaton associated to the formulæ a U (Xb R c). To construct the Büchi automaton equivalent to X(a U (Xb R c)), we add a new state s_{new} and for each transition tr starting from the initial state s^0_φ to a state s, a transition from s_{new} to s with the same label. Finally, we add the state s_{init} that we consider as initial and we connect s_{init} to s_{new} with a transition labeled with the true label.

Theorem 4: Let ψ = Xφ, A_φ be the Büchi automaton equivalent to φ and A_ψ be the LTL automaton built according to Definition 8. Then, Words(Xφ) $= \mathcal{L}_\omega(A_\psi)$.

D. Büchi automata for φ R Θ formula

The formulæ φ R Θ informally means that Θ is true until φ becomes true, or Θ is true forever. Thus, the construction of a Büchi automaton for φ R Θ can be done by construction the Büchi automaton associated to the fact that Θ is true until φ becomes true and the construction of a Büchi automaton associated to the fact that Θ is true forever. Finally, make the union between the two constructed Büchi automata. Consequently, to build the Büchi automaton for φ R Θ, we need to add two new states s_i and s_f to the set of states of the automaton A_φ. s_i becomes the single initial state of the resulting automaton and s_f is added to set of final states of the resulting automaton. The following transitions are added to the set of transitions of the resulting automaton:

- Transitions $s_i \xrightarrow{p \wedge \Theta} s$ if and only if and only if $s^0 \xrightarrow{p} s$ where s^0 is the initial state of A_φ.
- A loop over the added state s_i labeled with the propositional formula Θ
- A loop over the added state s_f labeled with the propositional formula Θ
- A transition $s_i \xrightarrow{\Theta} s_f$

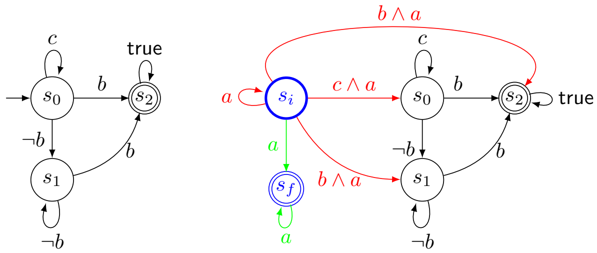
All other transitions of A_φ, as well as the accept states, remain unchanged.

Definition 9 (φ R Θ automaton): Let Θ be a propositional formula and φ be an LTL flat formulæ. Let $A_\varphi = (S_\varphi, s^0_\varphi, F_\varphi, \Sigma, \delta_\varphi)$ be the automaton associated to φ. The **automaton** $A_\psi = (S_\psi, s^0_\psi, F_\psi, \Sigma, \delta_\psi)$ associated to $\psi = \varphi$ R Θ is defined as follows:

- $S_\psi = \{s_i, s_f\} \cup S_\varphi$
- $s^0_\psi = s_i$, $F_\psi = F_\varphi \cup \{s_f\}$
- The transition function δ is defined as follows:

$$\delta_\psi(s,p) = \begin{cases} \delta_\varphi(s,p) \text{ if } s \in S_\varphi \text{ } (A_\varphi \text{ transitions}) \\ \delta_\varphi(s^0_\varphi, p\prime) \text{ if } s = s_i \text{ and } p = \Theta \wedge p\prime \\ (\text{Connection } s_i \text{ to initial state of } A_\varphi) \\ \{s_i, s_f\} \text{ if } s = s_i \text{ and } p = \Theta \\ (\text{Loop over } s_i \text{ or connection } s_i \text{ to } s_f) \\ \{s_f\} \text{ if } s = s_f \text{ and } p = \Theta \\ (\text{Loop over } s_f) \end{cases}$$

Example 3: Figure 3 illustrates the composition definition of φ R Θ. Figure 3a shows the Büchi automaton associated to the formulæ c U $\Diamond b$. To construct the Büchi automaton associated to the LTL formulæ $((c$ U $\Diamond b)$ R $a)$, we add a state s_i that we consider as the only initial state and a state s_f that we consider as a final state. We add a loop labeled with the atomic proposition a over the two added states. Finally, for each transition outgoing from the initial state of the automaton φ with label l and goes to state s, we add a transition from the added state s_i to the state s with a label $l \wedge a$. We also add a transition labeled with a from the state s_i to the state s_f.



(a) c U $\Diamond b$ (b) $(c$ U $\Diamond b)$ R a

Figure 3. Example of composition: φ R Θ

Theorem 5: Let $\psi = \varphi$ R Θ, A_φ be the Büchi automaton equivalent to φ and A_ψ be the LTL automaton built according to Definition 9. Then, Words(φ R Θ) $= \mathcal{L}_\omega(A_\psi)$.

E. Büchi for $\Box\Theta$ formula

The Büchi automaton associated to formulæ $\Box\Theta$ is obtained by creating one state s_0 and a loop over s_0 labeling with Θ.

Definition 10 ($\Box\varphi$ automaton): Let Θ be an propositional formulæ. The automaton associated to $\Box\Theta$ is defined as $A_{\Box\Theta} = (\{s_0\}, s_0, \{s_0\}, \mathsf{Prop}, \delta_{\Box\Theta})$ where $\delta_{\Box\Theta}$ is defined as follows: $\delta_{\Box\Theta}(s_0, \Theta) = \{s_0\}$

V. Our algorithm and its implementation

Our algorithm to build Büchi automata from FLTL formula is compositional in the sense that the final Büchi automaton is obtained by developing a sub-automaton for each subformulæ of the principal formulæ . Hence, the basic idea for developing the final automaton for a FLTL formulæ φ is to explore the formulæ φ in a preorder traversal. That is to say, we visit the root operator of φ first, then recursively do a preorder traversal of the left sub-formula, followed by a recursive preorder traversal of the right formulæ . Algorithm 1 allows us to build a Büchi automaton for a positive FLTL formula φ and uses the following five functions:

- BuchiProp(Θ): takes as input a propositional formula Θ and returns the automaton as defined in Definition 6 (Section IV);
- BuchiNext(BA): takes as input an Büchi automaton BA and returns a Büchi automaton defined according to Definition 8 (Section IV);
- BuchiEventuelly(BA): takes as input an Büchi automaton BA and returns a Büchi automaton defined according to Definition 7 (Section IV);
- BuchiBinary(op, $\mathsf{BA_l}$, $\mathsf{BA_r}$): that takes as input $\wedge$ or $\vee$ operator and two Büchi automata BA_l and BA_r and returns a Büchi automaton defined according to definitions of $\wedge$ and $\vee$ given in Section II;
- BuchiUntil(Θ, BA): that takes as input a propositional formula Θ and a Büchi automaton BA and returns the automaton as defined in Definition 7 (Section IV);
- BuchiRelease(Θ, BA) that takes as input a propositional formula Θ and a Büchi automaton BA and returns the automaton as defined in Definition 9 (Section IV).
- BuchiAlways(Θ): takes as input a propositional formula Θ and returns the automaton as defined in Definition 10 (Section IV);

Algorithm 1: Generating Büchi automata: **GenerateBA**(φ) for a FLTL formula

Name : GenerateBA
Input: a positive FLTL formulæ φ
Output: a Büchi automaton A;

if φ `instance of` U **then**
  `return BuchiUntil(Left` (φ), `GenerateBA(right` (φ)));
else if φ `instance of` R **then**
  `return BuchiRelease(right` (φ), `GenerateBA(Left` (φ)));
else if φ `instance of` X **then**
  `return BuchiNext(GenerateBA(right` (φ)));
else if φ `instance of` $\Box$ **then**
  `return BuchiAlways`(φ);
else if φ `instance of` $\Diamond$ **then**
  `return BuchiEventuelly(GenerateBA(right` (φ)));
else if *(*φ `instance of` $\vee$*) or (*φ `instance of` $\wedge$*)* **then**
  if `isPropositionnal` *(*`Left` *(*φ*)) and* `isPropositionnal` *(*`right` *(*φ*)* **then**
    `return BuchiProp`(φ) ;
  else if `isPropositionnal` *(*`Left` *(*φ*))* **then**
    `return BuchiBinary(BuchiProp(Left` (φ)),`GenerateBA(right` (φ)) ;
  else if `isPropositionnal` *(*`right` *(*φ*))* **then**
    `return BuchiBinary(GenerateBA(Left` (φ),`BuchiProp(right` (φ))) ;
  else
    `return BuchiBinary(BuchiProp(Left` (φ)),`BuchiProp(right` (φ))) ;

The proposed translation algorithm is very efficient where we can translate any FLTL formula φ of length n in time $O(n)$ with $O(n)$ states. The trick is to eliminate from our translation each step that could be exponential. As Büchi automata complementation is exponential [11][12], our transformation prohibit the use of complement Büchi automata operation and requires to use only LTL formula with negation pushed to atomic propsitions.

Theorem 6: For any FLTL formulæ $\varphi \in \mathcal{L}_f$, there exists an Büchi automaton A_φ with $|A_\varphi| = O(|\varphi|)$ and if A_ψ is the Büchi automaton generated by Algorithm 1, then: $Words(\psi) = \mathcal{L}_\omega(A_\psi)$.

We implemented our algorithm within the Graphical Tool for Omega-Automata and Logics (GOAL) tool that is an adequate graphical tool for defining and manipulating common variants of omega-automata, in particular Büchi automata, and temporal logic formula [13]. GOAL supports the translation of temporal formula such as Quantified Propositional Temporal Logic (QPTL) into Büchi automata where many well-known translation algorithms are implemented. It also provides language equivalence between two Büchi automata, automata

 ISBN: 978-1-61208-752-8

TABLE I. BENCHMARK FORMULA FOUND IN [14]

Formula	$\in \mathcal{L}_{FLTL}$
p U (q U □r)	yes
p U (q ∧ X(r U s))	yes
p U (q ∧ X(r ∧ (◇(s ∧X(◇(t ∧ X(◇(u ∧ X◇v))))))))	yes
◇(p ∧ X□q)	yes
◇(p ∧ X(q ∧ X◇r))	yes
◇(q ∧ X(p U r))	yes
(◇□q) ∨ (◇□p)	yes
◇(p ∧ X◇(q ∧ X◇(r ∧ X◇s)))	yes
□◇p ∧ □◇q ∧ □◇r ∧ □◇s ∧ □◇t	yes
(p U q U r) ∨ (q U r U p) ∨ (r U p U q)	yes
□(p → (q U (□r ∨ □s)))	no
□(p → (q U r))	no

TABLE II. BENCHMARK FORMULA FOUND IN [15][16]

Formula	∈	Formula	∈
p U q	yes	¬ □(p → X(q R r))	yes
p U (q U r)	yes	¬ (□◇p ∨ ◇□q)	yes
¬ (p U (q U r))	no	◇p ∧ ◇¬ p	yes
□◇p → □◇q	yes	(□(q ∨ □◇p) ∧ □(r ∨ □◇¬p)) ∨ □q ∨ □r	no
¬ (◇◇p ↔◇p)	yes	(□(q ∨ ◇□p) ∧ □(r ∨ ◇□¬p)) ∨ □q ∨ □r	no
¬ (□◇p → □◇q)	yes	¬ ((□(q ∨ □◇p) ∧ □(r ∨ □◇¬p)) ∨ □q ∨ □r)	yes
¬ (□◇p ↔ □◇q)	yes	¬((□(q ∨ ◇□p) ∧ □(r ∨ ◇□¬p)) ∨ □q ∨ □r)	yes
p R (p ∨ q)	yes	□(q ∨ X□p) ∧ □(r ∨ X□¬ p))	no
(Xp U Xq) ∨ ¬ X(p U q)	yes	□(q ∨ (Xp ∧ X¬ p))	no
(Xp U q) ∨ ¬ X(p U (p ∧ q))	yes	(p U p) ∨ (q U p)	yes
□(p → ◇q) ∧ ((Xp U q) ∨ ¬ X(p U (p ∧ q)))	no	◇p U □q	no
□(p → ◇q) ∧ ((Xp U Xq) ∨ ¬ X(p U p))	no	□p U q	no
□(p → ◇q)	no	□(◇p ∧ ◇q)	yes
(Xq ∧ r) R X(((s U p) R r) U (s R r))	no		

TABLE III. COVERAGE DWYER'S PATTERNS/SCOPES BY OUR LTL FRAGMENT

Scope/Pattern	Globally	Before r	After q	Between q and r	After q until r
Absence	yes	yes	yes	yes	yes
Universality	yes	yes	yes	yes	no
Existence	no	no	yes	yes	yes
Precedence	yes	yes	yes	no	yes
Response	yes	yes	no	no	no
s, t precedes p	yes	yes	yes	no	no
p precedes s, t	yes	yes	yes	no	no
p responds s t	no	yes	no	no	no
s, t responds p	no	yes	no	no	no
s, t without z responds to p	no	yes	no	no	no

complementation, automata union, automata intersection and emptiness algorithms. It has extensions covering common translation algorithms (*e.g.*, LTL2BA [8], Tableau algorithm, LTL2AUT, *etc.*). As the recent implementation of GOAL is based on the Java Plugin Framework, it can be properly extended by new plug-ins, providing new functionalities that are loaded at run-time. We implemented our composition algorithm within an independent plug-in. The automata generated by our algorithm are simplified by several simplification methods (*e.g.*, simulation, Delayed simulation, Faired simulation, reducing unreachable/dead states) by taking advantage from GOAL tool which implements all these methods.

VI. COVERAGE AVERAGE OF FLTL FRAGMENT

In this section, we present the experiments that we conducted to show the coverage average of our fragment *FLTL* formula. Three sets LTL formula which commonly considered in the literature are performed. The experiments on the one hand, emphasis the performance of our implementation algorithm and, on the other, demonstrates that a wide range of LTL formula can be covered by our approach and translating polynomially and properly using our GOAL plug-in. The process we applied for each formula φ in our experiments can be summarized as follows:

1) Checking whether φ belongs to *FLTL* fragment by building the finite syntax tree of φ.
2) Using the well-known algorithm LTL2BA to generate a Büchi automaton equivalent to φ (called A_1)
3) Using our GOAL plugin to generate the Büchi automaton A_2 according to rules defined in our algorithm (*i.e.*, Algorithm 1)
4) Runing the GOAL Büchi automata equivalence to check the equivalence between A_1 and A_2.

The first set contains 12 formula and can be found in [14]. The experiments for this set show that only two formula do not belong to our grammar as shown in Table I. The coverage average for this set is then 83.334%.

The second set contains 27 formula and can be found in [15][16]. The results show that the *FLTL* fragment fails to express only 11 formula as shown in Table II. The coverage average for this set is then 59.259%.

The third set contains 50 formula and can be found in [17]. Indeed, the authors in [17] have proposed a pattern-based approach which uses specification patterns that, at a higher abstraction level, capture recurring temporal properties. The main idea is that a temporal property is a combination of one **pattern** and one **scope**. A scope is the part of the system execution path over which a pattern holds. For more details about patterns and scopes can be found in [17]. They proved that the patterns dramatically simplify the specification of temporal properties, with a fairly complete coverage where they collected hundreds of specifications and they observed that 92% of them fall into this small set of patterns/scopes. A translational semantics have been proposed to Dwyer's properties by mapping each pattern/scope combination to a corresponding LTL formula. As Dwyer's and *al.* propose 5 scopes and 10 patterns, the total number of involved LTL formula is then 50. The results of the comparison are given in Table III and show that our LTL fragment covers 27 formula from 50 formula associating by Dwyer to scopes/patterns. The coverage average for this set is then 54%.

The covering average of each set is accepted and slightly high. This shows that our fragment covers more than 65.531%, which is considered very good enough due to the importance of LTL formalism in modeling area. Such a result could be a promising direction to explore LTL-based model checking techniques in which system properties are first expressed in LTL formula then converted into Büchi automata.

VII. RELATED WORK

Translation from LTL formula to Büchi automata has been extensively studied in the literature. Authors in [1][18] constructed Büchi automata whoses states are sets of subformula of the considered LTL formula. This translation is of order $2^{O(n)}$ where n is the length of the LTL formula in input. [19] proposed to build Büchi automata by a bottom-up traversal through the syntax tree of the considered LTL formula. This translation has been proved in order of $2^{O(nlog(n))}$. [20] presented an efficient translation by means of alternating ω-automata. The translation from LTL formula to alternating ω automata is linear in terms of the length of the considered LTL formula, but the translation of the resulting alternating ω-automaton to the target Büchi automata is exponential. [21]-

 ISBN: 978-1-61208-752-8

[22] proposed on-the-fly translation of so-called generalized Büchi automata (Büchi automata with multiple acceptance conditions) which then linearly converted into Büchi automata.

There are several fragments of LTL that have been proposed in the literature. [3] has proved that converting any formula in which the only allowed modality is the until operator U or the only allowed modality is X or $\diamondsuit$ to Büchi automata is PSPACE. The formula that uses only the $\diamondsuit$ operator is coNP-Complete. The formula that uses only the X operator is coNP-Complete [23]. The formula that uses only the $\diamondsuit$ operator in the form $\square\diamondsuit$ is co-NPComplete [24]. In [23], the authors used the term Flat LTL to express formula that use the U operator whose the left-hand side does not contain any temporal combinator, but the right-side can contain only formula with the U operator (or its negation). Translation from this fragement to Büchi automata has been proved NP-Complete. Several simple cases with a lower worst-case complexity are handled in [23][24].

VIII. Conclusion and future work

This paper presented a compositional algorithm for generating Büchi automata from a fragment of LTL logic. First, we proposed the grammar of this fragment and then built for each formulæ φ, its equivalent Büchi automata. Second, we showed theoretically how to compositionnally build from Büchi automata associated to each sub-formulæ, the Büchi automaton of the target formulæ. Third, we implemented our approach in GOAL tool as a plugin and showed the complexity and the correctness of our Büchi automata generation method. Fourth, we demonstrated the interest of our method by computing coverage average of the fragment FLTL using three sets of well-known LTL formulas as benchmarks.

Several research lines can be continued from the present work. First, some temporal operators such as always, precedes or since are not considered in this paper, as an immediate perspective, we will study how to include these operators in our LTL fragment. Second, it will be interesting to study whether our fragment LTL is minimalist and whether there is possibility to more expand it by identifying what makes it smallest. A good direction for this point is to study whether there is a subset of Dwyer's pattern/scope from which all other patterns/scopes can be deduced. Third, it would be interesting to connect the proposed language to usual model checking tools.

Acknowledgment

This project has been jointly funded with the support of the National Council for Scientific Research in Lebanon CNRS-L and Lebanese University.

References

[1] O. Lichtenstein and A. Pnueli, "Checking that finite state concurrent programs satisfy their linear specification," in Proceedings of the 12th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages. New York, NY, USA: ACM, 1985, pp. 97–107.

[2] C. Baier and J. Katoen, Principles of Model Checking (Representation and Mind Series). The MIT Press, 2008.

[3] A. Sistla and E. Clarke, "The complexity of propositional linear temporal logics," J. ACM, vol. 32, no. 3, july 1985, pp. 733–749.

[4] R. Hierons and al., "Using formal specifications to support testing," ACM Comput. Surv., vol. 41, February 2009, pp. 9:1–9:76.

[5] S. Gnesi, D. Latella, M. Massink, V. Moruzzi, and I. Pisa, "Formal test-case generation for UML statecharts," in Proc. 9th IEEE Int. Conf. on Engineering of Complex Computer Systems. IEEE Computer Society, 2004, pp. 75–84.

[6] E. Clarke, O. Grumberg, and K. Hamaguchi, "Another look at LTL model checking," in Formal methods in system design. Springer-Verlag, 1994, pp. 415–427.

[7] M. Vardi, "Branching vs. linear time: Final showdown," in Proceedings of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems. London: Springer, 2001, pp. 1–22.

[8] P. Gastin and D. Oddoux, "Fast LTL to Büchi automata translation," in Proceedings of the 13th International Conference on Computer Aided Verification (CAV'01), ser. LNCS, vol. 2102. Paris, France: Springer, jully 2001, pp. 53–65.

[9] V. King, O. Kupferman, and M. Vardi, On the Complexity of Parity Word Automata. Springer Berlin Heidelberg, 2001, pp. 276–286.

[10] E. A. Emerson, "Handbook of theoretical computer science (vol. b)," J. van Leeuwen, Ed. Cambridge, MA, USA: MIT Press, 1990, ch. Temporal and Modal Logic, pp. 995–1072.

[11] S. Safra, "On the complexity of omega-automata," in 29th Annual Symposium on Foundations of Computer Science, White Plains, New York, USA, 24-26 October 1988, 1988, pp. 319–327.

[12] A. Sistla, M. Vardi, and P. Wolper, "The complementation problem for büchi automata with applications to temporal logic," in Automata, Languages and Programming. Springer Berlin Heidelberg, 1985, pp. 465–474.

[13] Y.-K. Tsay, Y.-F. Chen, M.-H. Tsai, K.-N. Wu, and W.-C. Chan, "Goal: A graphical tool for manipulating büchi automata and temporal formulae," in Tools and Algorithms for the Construction and Analysis of Systems. Springer Berlin Heidelberg, 2007, pp. 466–471.

[14] K. E. and G. Holzmann, "Optimizing Büchi automata." Springer, 2000, pp. 153–167.

[15] M. Daniele, F. Giunchiglia, and M. Vardi, "Improved automata generation for linear temporal logic," in In 11th International Conference on Computer Aided Verification, ser. CAV '99. London, UK: Springer, 1999, pp. 249–260.

[16] F. Somenzi and R. Bloem, "Efficient büchi automata from ltl formulae," in Computer Aided Verification, E. A. Emerson and A. P. Sistla, Eds. Berlin, Heidelberg: Springer, 2000, pp. 248–263.

[17] M. Dwyer, G. Avrunin, and J. Corbett, "Patterns in property specifications for finite-state verification," in Proceedings of the 21st International Conference on Software Programming, 1999, pp. 411–420.

[18] P. Wolper, "On the relation of programs and computations to models of temporal logic," in Temporal Logic in Specification, B. Banieqbal, H. Barringer, and A. Pnueli, Eds. Springer Berlin Heidelberg, 1989, pp. 75–123.

[19] G. G. de Jong, "An automata theoretic approach to temporal logic," in Computer Aided Verification, K. G. Larsen and A. Skou, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1992, pp. 477–487.

[20] M. Y. Vardi, An automata-theoretic approach to linear temporal logic. Berlin, Heidelberg: Springer, 1996, pp. 238–266.

[21] J.-M. Couvreur, "On-the-fly verification of linear temporal logic," in FM'99 — Formal Methods, J. M. Wing, J. Woodcock, and J. Davies, Eds. Springer, 1999, pp. 253–271.

[22] S. Schwoon and J. Esparza, "A note on on-the-fly verification algorithms," in Tools and Algorithms for the Construction and Analysis of Systems, N. Halbwachs and L. D. Zuck, Eds. Springer Berlin Heidelberg, 2005, pp. 174–190.

[23] S. Demri and P. Schnoebelen, "The complexity of propositional linear temporal logics in simple cases," Information and Computation, vol. 174, no. 1, 2002, pp. 84 – 103.

[24] E. A. Emerson and C.-L. Lei, "Modalities for model checking: branching time logic strikes back," Science of Computer Programming, vol. 8, no. 3, 1987, pp. 275 – 306.