On the Relationship between LTL Normal Forms and Büchi Automata

Jianwen Li[1], Geguang Pu[1⋆], Lijun Zhang[2],
Zheng Wang[1], Jifeng He[1], and Kim G. Larsen[3]

[1] Software Engineering Institute East China Normal University, P. R.China
[2] State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences
[3] Computer Science, Aalborg University, Denmark

Abstract. In this paper, we revisit the problem of translating LTL formulas to Büchi automata. We first translate the given LTL formula into a special *disjuctive-normal form* (DNF). The formula will be part of the state, and its DNF normal form specifies the atomic properties that should hold immediately (labels of the transitions) and the *formula* that should hold afterwards (the corresponding successor state). If the given formula is Until-free or Release-free, the Büchi automaton can be obtained directly in this manner. For a general formula, the construction is more involved: an additional component will be needed for each formula that helps us to identify the set of accepting states. Notably, our construction is an on-the-fly construction, which starts with the given formula and explores successor states according to the normal forms. We implement our construction and compare the tool with SPOT [3]. The comparision results are very encouraging and show our construction is quite innovative.

1 Introduction

Translating Linear Temporal Logic (LTL) formulas to their equivalent automata (usually Büchi automata) has been studied for nearly thirty years. This translation plays a key role in the automata-based model checking [14]: here the automaton of the negation of the LTL property is first constructed, then the verification process is reduced to the emptiness problem of the product automaton (from the property automaton and the system model). Gerth et al. [6] proposed an on-the-fly construction approach to generating Büchi automata from LTL formulas, in which counterexamples can be detected even only a part of the property automaton is generated. They called it a tableau construction approach, which became widely used and many subsequent works [11, 7, 2, 4, 1] for optimizing the automata under construction are based on it.

In this paper, we propose a novel construction by making use of the notion of *disjuctive-normal forms* (DNF). For an LTL formula φ, our DNF normal form is an equivalent formula of the form $\bigvee_i(\alpha_i \wedge X\varphi_i)$ where α_i is a finite conjunction of literals (atomic propositions or their negations), and φ_i is a conjunctive LTL formula such that the root operator of it is not a disjunction. We show that any LTL formula can be

⋆ Corresponding author.

transformed into an equivalent DNF normal form, and refer to $\alpha_i \wedge X\varphi_i$ as a clause of φ. In this way any given LTL formula induces a labelled transition system (LTS): states correspond to formulas, and we assign a transition from φ to φ_i labelled with α_i, if $\alpha_i \wedge X\varphi_i$ is a clause of φ.

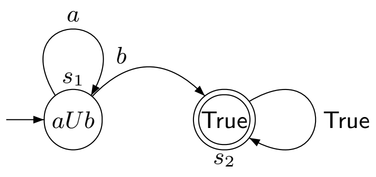

Fig. 1. The Büchi automaton for aUb

This LTS is the starting point of our construction. Firsly, for Until-free (or Release-free) formulas, the Büchi automaton can be obtained directly by equipping the above LTS with the set of accepting states, which is illustrated as follows. Consider the formula aUb, whose DNF form is $(b \wedge X(\mathsf{True})) \vee (a \wedge X(aUb))$. The corresponding Büchi automaton for aUb is shown in Figure 1 where nodes aUb and True represent formulas aUb and True respectively. The transitions are self-explained. By semantics, we know that if the run ξ satisfies a Release-free formula φ, then there must be a finite satisfying prefix η of ξ such that any paths starting with η satisfy φ as well. Thus, for this class of formulas, the state corresponding to the formula True is considered as the single accepting state. The Until-free formulas can be treated in a similar way by taking the set of all states as accepting.

The main contribution of the paper is to extend the above construction to general formulas. As an example we consider the formula $\psi = G(aUb)$, which has the normal form $(b \wedge X\psi) \vee (a \wedge X(aUb \wedge \psi))$. Note here the formula True will be even not reachable. The most challenging part of the construction will then be identification of the set of accepting states. For this purpose, we identify first subformulas that will be reached infinitely often, which we call looping formulas. Only some of the looping formulas contribute to the set of accepting states. These formulas will be the key to our construction: we characterize a set of atomic propositions for each formula, referred to as the *obligation set*. The set contains various obligations, each represented as a set of literals, that must occur infinitely often to make the given formula satisfiable. In our construction, we add an additional component to the states to keep track of the obligations, and then define accepting states based on it – an illustrating example can be found in Section 2.

Our construction for general formula works on-the-fly: it starts with the given formula and explore successor states according to the normal forms. We implement our construction and compare the tool with SPOT [3]. The results show our tool competes with, and sometimes outperforms SPOT under the benchmarks tested, which is encouraging as SPOT is the state-of-the-art tool that has been highly optimized.

Related Work. As we know, there are two main approaches of Büchi automata construction from LTL formulas. The first approach generates the alternating automaton from the LTL formula and then translates it to the equivalent Büchi automaton [13]. Gastin et al. [5] proposed a variant of this construction in 2001, which first translates the very weak alternating co-Büchi automaton to generalised automaton with accepting transitions which is then translated into Büchi automaton. In particular, the experiments show that their algorithm outperforms the others if the formulas under construction are restricted on fairness conditions. Recently Babiak et al. [1] proposed some optimization strategies based on the work [5].

The second approach was proposed in 1995 by Gerth et al. [6], which is called the *tableau* construction. This approach can generate the automata from LTL on-the-fly, which is widely used in the verification tools for acceleration of the automata-based verification process. Introducing the (state-based) *Generalized Büchi Automata* (GBA) is the important feature for the tableau construction. Daniele et al. [2] improved the tableau construction by some simple syntactic techniques. Giannakopoulou and Lerda [7] proposed another construction approach that uses the transition-based Generalized Büchi automaton (TGBA). Some optimization techniques [4, 11] have been proposed to reduce the size of the generated automata. For instance, Etessami and Holzmann [4] described the optimization techniques including proof theoretic reductions (formulas rewritten), core algorithm tightening and the automata theoretic reductions (simulation based).

Organization of the paper. Section 2 illustrates our approach by a running example. Section 3 introduces preliminaries of Büchi automata and LTL formulas; Section 4 specifies the proposed *DNF-based* construction; Section 5 compares the experimental results from our tool and SPOT. Section 6 discusses how our approach is related to the tableau construction. Section 7 concludes the paper. Proofs can be found in the report [8].

2 A Running Example

We consider the formula $\varphi_1 = \mathrm{G}(bUc \wedge dUe)$ as our running example. The DNF form of φ_1 is given by:

$$\varphi_1 = (c \wedge e \wedge X(\varphi_1)) \vee (b \wedge e \wedge X(\varphi_2)) \vee (c \wedge d \wedge X(\varphi_3)) \vee (b \wedge d \wedge X(\varphi_4)$$

where $\varphi_2 = bUc \wedge G(bUc \wedge dUe)$, $\varphi_3 = dUe \wedge G(bUc \wedge dUe)$, $\varphi_4 = bUc \wedge dUe \wedge G(bUc \wedge dUe)$. It is easy to check that the above DNF form is indeed equivalent to formula φ_1. Interestingly, we note that $\varphi_1, \varphi_2, \varphi_3, \varphi_4$ all have the same DNF form above.

The corresponding Büchi automaton for φ_1 is depicted in Fig. 2. We can see that there are four states in the generated automata, corresponding to the four formulas $\varphi_i(i = 1, 2, 3, 4)$. The state corresponding to the formula φ_1 is also the initial state. The transition relation is obtained by observing the DNF forms: for instance we have a self-loop for state s_1 with label $c \wedge e$. If we observe the normal form of φ_1, we can see that there is a term $(c \wedge e \wedge X(\varphi_1))$, where there is a conjunction of two terms $c \wedge e$ and

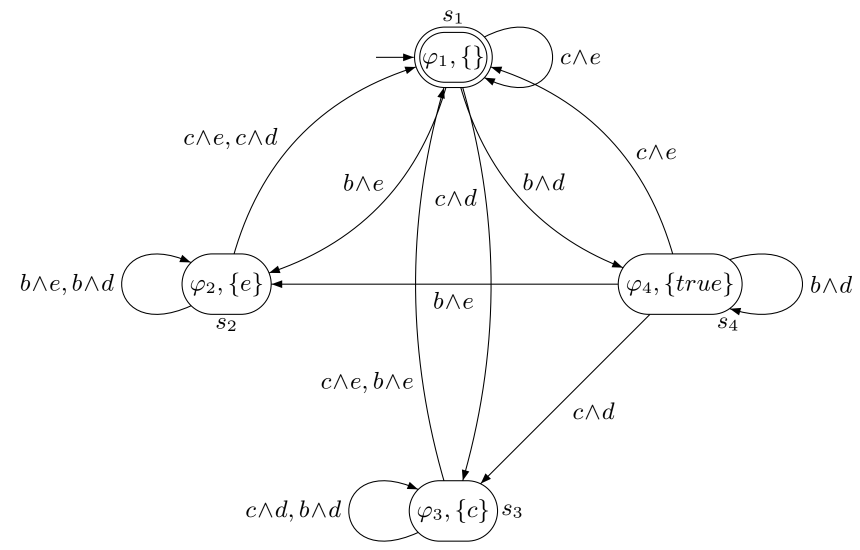

Fig. 2. The Büchi automaton for the formula φ_1.

$X(\varphi_1)$, and φ_1 in X operator corresponds to the node s_1 and $c \wedge e$ corresponds to the loop edge for s_1.

Thus, the *disjunctive-normal form* of the formula has a very close relation with the generated automaton. The most difficult part is to determine the set of accepting states of the automaton. We give thus here a brief description of several notions introduced for this purpose in our running example. The four of all the formulas $\varphi_i (i = 1, 2, 3, 4)$ have the same *obligation set*, i.e. $OS_{\varphi_i} = \{\{c, e\}\}$, which may vary for different formulas. In our construction, every *obligation* in the *obligation set* of each formula identities the properties needed to be satisfied infinitely if the formula is satisfiable. For example, the formulas $\varphi_i (i = 1, 2, 3, 4)$ are satisfied if and only if all properties in the obligation $\{c, e\}$ are met infinitely according to our framework. Then, a state consists of a formula and the *process set*, which records all the properties that have been met so far. For simplicity, we initialize the *process set* P_1 of the initial state s_1 with the empty set. For the state s_2, the corresponding process set $P_2 = \{e\}$ is obtained by taking the union of P_1 and the label $\{b, e\}$ from s_1. The label b will be omitted as it is not contained in the obligation. Similarly one can conclude $P_3 = \{c\}$ and $P_4 = \{true\}$: here the property $true$ implies no property has been met so far. When there is more than one property in the *process set*, the $\{true\}$ can be erased, such as that in state s_3. Moreover, the *process set* in a state will be reset to empty if it includes one *obligation* in the formula's *obligation set*. For instance, the transition in the figure $s_2 \xrightarrow{c \wedge d} s_1$ is due to that $P_1' = P_2 \cup \{c\} = \{c, e\}$, which is actually in OS_{φ_1}. So P_1' is reset to the empty set. One can also see the same rule when the transitions $s_2 \xrightarrow{c \wedge e} s_1$, $s_4 \xrightarrow{c \wedge e} s_1$, $s_3 \xrightarrow{b \wedge e} s_1$ occur.

Through the paper, we will go back to this example again when we explain our construction approach.

3 Büchi Automaton, LTL Formulas

3.1 Büchi Automaton

A Büchi automaton is a tuple $\mathcal{A} = (S, \Sigma, \delta, S_0, F)$, where S is a finite set of states, Σ is a finite set of alphabet symbols , $\delta : S \times \Sigma \to 2^S$ is the transition relation, S_0 is a set of initial states, and $F \subseteq S$ is a set of accepting states of $\mathcal{A}$.

We use $w, w_0 \in \Sigma$ to denote alphabets in Σ, and $\eta, \eta_0 \in \Sigma^*$ to denote finite sequences. A *run* $\xi = w_0 w_1 w_2 \dots$ is an infinite sequence over Σ^ω. For ξ and $k \geq 1$ we use $\xi^k = w_0 w_1 \dots w_{k-1}$ to denote the prefix of ξ up to its kth element (the $k+1$th element is not included) as well as ξ_k to denote the suffix of $w_k w_{k+1} \dots$ from its $(k+1)$th element (the $k+1$th element is included). Thus, $\xi = \xi^k \xi_k$. For notational convenience we write $\xi_0 = \xi$ and $\xi^0 = \varepsilon$ (ε is the empty string). The run ξ is accepting if it runs across one of the states in F infinitely often.

3.2 Linear Temporal Logic

We recall the linear temporal logic (LTL) which is widely used as a specification language to describe the properties of reactive systems. Assume AP is a set of atomic properties, then the syntax of LTL formulas is defined by:

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

where $a \in AP$, φ is an LTL formula. We say φ is a *literal* if it is a proposition or its negation. In this paper we use lower case letters to denote atomic properties and α, β, γ to denote propositional formulas (without temporal operators), and use φ, ψ, ϑ, μ, ν and λ to denote LTL formulas.

Note that w.l.o.g. we are considering LTL formulas in negative normal form (NNF) – all negations are pushed down to literal level. LTL formulas are interpreted on infinite sequences (correspond to runs of the automata) $\xi \in \Sigma^\omega$ with $\Sigma = 2^{AP}$. The Boolean connective case is trivial, and the semantics of temporal operators is given by:

- $\xi \models \varphi_1 \ U \ \varphi_2$ iff there exists $i \geqslant 0$ such that $\xi_i \models \varphi_2$ and for all $0 \leqslant j < i, \xi_j \models \varphi_1$;
- $\xi \models \varphi_1 \ R \, \varphi_2$ iff either $\xi_i \models \varphi_2$ for all $i \geq 0$, or there exists $i \geq 0$ with $\xi_i \models \varphi_1 \wedge \varphi_2$ and $\xi_j \models \varphi_2$ for all $0 \leq j < i$;
- $\xi \models X \ \varphi$ iff $\xi_i \models \varphi$.

According to the LTL semantics, it holds $\varphi R \psi = \neg(\neg\varphi U \neg\varphi)$. We use the usual abbreviations $\mathsf{True} = a \vee \neg a$, $Fa = \mathsf{True} U a$ and $Ga = \mathsf{False} R a$.

Notations. Let φ be a formula written in *conjunctive form* $\varphi = \bigwedge_{i \in I} \varphi_i$ such that the root operator of φ_i is not a conjunctive: then we define the conjunctive formula set as $CF(\varphi) := \{\varphi_i \mid i \in I\}$. When φ does not include a conjunctive as a root operator, $CF(\varphi)$ only includes φ itself. For technical reasons, we assume that $CF(\mathsf{True}) = \emptyset$.

4 *DNF-based* Büchi Automaton Construction

Our goal of this section is to construct the Büchi automaton $\mathcal{A}_\lambda$ for λ. We first present the disjunctive normal forms, and then establish a few simple properties of general formulas that shall shed insights on the construction for the $Release$-free ($Until$-free) formulas. We then define the labelled transition system for a formula. In the following three subsections we present the construction for $Release$-free ($Until$-free) and general formulas, respectively.

In the remaining of the paper, we fix λ as the input LTL formula. All formulas being considered will vary over the set $EF(\lambda)$, and AP will denote the set of all literals appearing in λ, and $\Sigma = 2^{AP}$. Moreover, we assume all atomic propositions in λ are *syntactically different*, but maintain their *semantically equivalence*. For example, the formula $aU(a \wedge b)$ will be identified by $a_1 U(a_2 \wedge b)$, where $a_1 \equiv a_2$. This will be used to track whether the atomic proposition are from the left part or right part of the until operator.

4.1 Disjunctive Normal Form

We introduce the notion of *disjunctive-normal form* for LTL formulas in the following.

Definition 1 (disjunctive-normal form). *A formula φ is in disjunctive-normal form (DNF) if it can be represented as $\varphi := \bigvee_i (\alpha_i \wedge X\varphi_i)$, where α_i is a finite conjunction of literals, and $\varphi_i = \bigwedge \varphi_{i_j}$ where φ_{i_j} is either a literal, or an Until, Next or Release formula.*

We say $\alpha_i \wedge X\varphi_i$ is a clause *of φ, and write $DNF(\varphi)$ to denote all of the clauses.*

As seen in the introduction and motivating example, DNF form plays a central role in our construction. Thus, we first discuss that any LTL formula φ can be transformed into an equivalent formula in DNF form. The transformation is done in two steps. The first step is according to the following rules:

Lemma 1. *1. $DNF(\alpha) = \{\alpha \wedge X(\mathsf{True})\}$ where α is a literal;*
2. $DNF(X\varphi) = \{\mathsf{True} \wedge X(\varphi)\}$;
3. $DNF(\varphi_1 U \varphi_2) = DNF(\varphi_2) \cup DNF(\varphi_1 \wedge X(\varphi_1 U \varphi_2))$;
4. $DNF(\varphi_1 R \varphi_2) = DNF(\varphi_1 \wedge \varphi_2) \cup DNF(\varphi_2 \wedge X(\varphi_1 R \varphi_2))$;
5. $DNF(\varphi_1 \vee \varphi_2) = DNF(\varphi_1) \cup DNF(\varphi_2)$;
6. $DNF(\varphi_1 \wedge \varphi_2) = \{(\alpha_1 \wedge \alpha_2) \wedge X(\psi_1 \wedge \psi_2) \mid \forall i = 1, 2.\, \alpha_i \wedge X(\psi_i) \in DNF(\varphi_i)\}$;

All of the rules above are self explained, following by the definition of DNF, distributive and the expansion laws. What remains is how to deal with the formulas in the *Next* operator: by definition, in a clause $\alpha_i \wedge X(\varphi_i)$ the root operators in φ_i cannot be disjunctions. The equivalence $X(\varphi_1 \vee \varphi_2) = X\varphi_1 \vee X\varphi_2$ can be applied repeatedly to move the disjunctions out of the *Next* operator. The distributive law of disjunction over conjunctions allows us to bring any formula into an equivalent DNF form:

Theorem 1. *Any LTL formula φ can be transformed into an equivalent formula in disjunctive-normal form.*

In our running example, we have $DNF(\varphi_1) = DNF(\varphi_2) = DNF(\varphi_3) = DNF(\varphi_4) = \{c \wedge e \wedge X(\varphi_1), b \wedge e \wedge X(\varphi_2), c \wedge d \wedge X(\varphi_3), b \wedge d \wedge X(\varphi_4)\}$. Below we discuss the set of formulas that can be reached from a given formula.

Definition 2 (Formula Expansion). *We write $\varphi \xrightarrow{\alpha} \psi$ iff there exists $\alpha \wedge X(\psi) \in DNF(\varphi)$. We say ψ is expandable from φ, written as $\varphi \hookrightarrow \psi$, if there exists a finite expansion $\varphi \xrightarrow{\alpha_1} \psi_1 \xrightarrow{\alpha_2} \psi_2 \xrightarrow{\alpha_3} \ldots \psi_n = \psi$. Let $EF(\varphi)$ denote the set of all formulas that can be expanded from φ.*

The following theorem points out that $|EF(\lambda)|$ is bounded:

Theorem 2. *For any formula λ, $|EF(\lambda)| \leq 2^n + 1$ where n denotes the number of subformulas of λ.*

4.2 Transition Systems for LTL Formulas

We first extend formula expansions to subset in Σ:

Definition 3. *For $\omega \in \Sigma$ and propositional formula α, $\omega \models \alpha$ is defined in the standard way: if α is a literal, $\omega \models \alpha$ iff $\alpha \in \omega$, and $\omega \models \alpha_1 \wedge \alpha_2$ iff $\omega \models \alpha_1$ and $\omega \models \alpha_2$, and $\omega \models \alpha_1 \vee \alpha_2$ iff $\omega \models \alpha_1$ or $\omega \models \alpha_2$.*

We write $\varphi \xrightarrow{\omega} \psi$ if $\varphi \xrightarrow{\alpha} \psi$ and $w \models \alpha$. For a word $\eta = \omega_0\omega_1..\omega_k$, we write $\varphi \xrightarrow{\eta} \psi$ iff $\varphi \xrightarrow{\omega_0} \psi_1 \xrightarrow{\omega_1} \psi_2 \xrightarrow{\omega_2} ..\psi_{k+1} = \psi$.

For a run $\xi \in \Sigma^\omega$, we write $\varphi \xrightarrow{\xi} \varphi$ iff ξ can be written as $\xi = \eta_0\eta_1\eta_2 \ldots$ such that η_i is a finite sequence, and $\varphi \xrightarrow{\eta_i} \varphi$ for all $i \geq 0$.

Below we provide a few interesting properties derived from our DNF normal forms.

Lemma 2. *Let ξ be a run and λ a formula. Then, for all $n \geq 1$, $\xi \models \lambda \Leftrightarrow \lambda \xrightarrow{\xi^n} \varphi \wedge \xi_n \models \varphi$.*

Essentially, $\xi \models \lambda$ is equivalent to that we can reach a formula φ along the prefix ξ^n such that the suffix ξ_n satisfies φ. The following corollary is a direct consequence of Lemma 2 and the fact that we have only finitely many formulas in $EF(\lambda)$:

Corollary 1. *If $\xi \models \lambda$, then there exists $n \geq 1$ such that $\lambda \xrightarrow{\xi^n} \varphi \wedge \xi_n \models \varphi \wedge \varphi \xrightarrow{\xi_n} \varphi$. On the other side, if $\lambda \xrightarrow{\xi^n} \varphi \wedge \xi_n \models \varphi \wedge \varphi \xrightarrow{\xi_n} \varphi$, then $\xi \models \lambda$.*

This corollary gives the hint that after a finite prefix we can focus on whether the suffix satisfies the *looping formula* φ, i.e,. those φ with $\varphi \hookrightarrow \varphi$. From Definition 2 and the expansion rules for LTL formulas, we have the following corollary:

Corollary 2. *If $\lambda \hookrightarrow \lambda$ holds and $\lambda \neq$ True, then there is at least one Until or Release formula in $CF(\lambda)$.*

As we described in previous, the elements in $EF(\lambda)$ and its corresponding DNF-normal forms naturally induce a labelled transition system, which can be defined as follows:

Definition 4 (LTS for λ). *The labelled transition system TS_λ generated from the formula λ is a tuple $\langle \Sigma, S, \delta, S_0 \rangle$: where $\Sigma = 2^{AP}$, $S = EF(\lambda)$, $S_0 = \{\lambda\}$ and δ is defined as follows: $\psi \in \delta(\varphi, \omega)$ iff $\varphi \xrightarrow{\omega} \psi$ holds, where $\varphi, \psi \in EF(\lambda)$ and $\omega \in \Sigma$.*

4.3 Büchi automata for Release/Until-free Formulas

The following lemma is a special instance of our central theorem 4. It states properties of accepting runs with respect to Release/Until-free formulas:

Lemma 3. *1. Assume λ is $Release$-free. Then, $\xi \models \lambda \Leftrightarrow \exists n \cdot \lambda \xrightarrow{\xi^n} \mathsf{True}$.*
2. Assume λ is $Until$-free. Then $\xi \models \lambda \Leftrightarrow \exists n, \varphi \cdot \lambda \xrightarrow{\xi^n} \varphi \wedge \varphi \xrightarrow{\xi_n} \varphi$.

Essentially, If λ is Release-free, we will reach True after finitely many steps; If λ is Until-free we will reach a looping formula after finitely many steps. The Büchi automaton for Release-free or Until-free formulas will be directly obtained by equipping the LTS with the set of accepting states:

Definition 5 ($\mathcal{A}_\lambda$ for Release/Until-free formulas). *For a Release/Until-free formula λ, we define the Büchi automaton $\mathcal{A}_\lambda = (S, \Sigma, \delta, S_0, F)$ where $TS_\lambda = \langle \Sigma, S, \delta, S_0 \rangle$. The set F is defined by: $F = \{\mathsf{True}\}$ if λ is Release-free while $F = S$ if λ is Until-free.*

Notably, True is the only accepting state for $\mathcal{A}_\lambda$ when λ is Release-free while all the states are accepting ones if it is Until-free.

Theorem 3 (Correctness and Complexity). *Assume λ is $Until$-free or $Release$-free. Then, for any sequence $\xi \in \Sigma^\omega$, it holds $\xi \models \lambda$ iff ξ is accepted by $\mathcal{A}_\lambda$. Moreover, $\mathcal{A}_\lambda$ has at most $2^n + 1$ states, where n is the number of subformulas in λ.*

Proof. The proof of the correctness is trivial according to Lemma 3: 1) if λ is Release-free, then every run ξ of $\mathcal{A}_\lambda$ can run across the True-state[4] infinitely often iff it satisfies $\exists n \geq 0 \cdot \lambda \xrightarrow{\xi^n} \mathsf{True}$, that is, $\xi \models \lambda$; 2) if λ is Until-free, then $\xi \models \lambda$ iff $\exists n, \varphi \cdot \lambda \xrightarrow{\xi^n} \varphi \wedge \varphi \xrightarrow{\xi_n} \varphi$, which will run across φ-state infinitely often so that is accepted by $\mathcal{A}_\lambda$ according to the construction.

The upper bound is a direct consequence of Theorem 2.

4.4 Central Theorem for General Formulas

In the previous section we have constructed Büchi automaton for Release-free or Until-free formulas, which is obtained by equipping the defined LTS with appropriate accepting states. For general formulas, this is however slightly involved. For instance, consider the LTS of the formula $\varphi = G(bUc \wedge dUe)$ in our running example: there are infinitely many runs starting from the initial state s_1, but which of them should be accepting? Indeed, it is not obvious how to identify the set of accepting states. In this section we present our central theorem for general formulas aiming at identifying the accepting runs.

Assume the run $\xi = \omega_0\omega_1\ldots$ satisfies the formula λ. We refer to $\lambda(= \varphi_0) \xrightarrow{\alpha_0} \varphi_1 \xrightarrow{\alpha_1} \varphi_2 \ldots$ as an expansion path from λ, which corresponds to a path in the LTS TS_λ, but labelled with propositional formulas. Obviously, $\xi \models \lambda$ implies that there exists an expansion path in TS_λ such that $\omega_i \models \alpha_i$ for all $i \geq 0$. As the set $EF(\lambda)$ is

[4] In this paper we use φ-state to denote the state representing the formula φ.

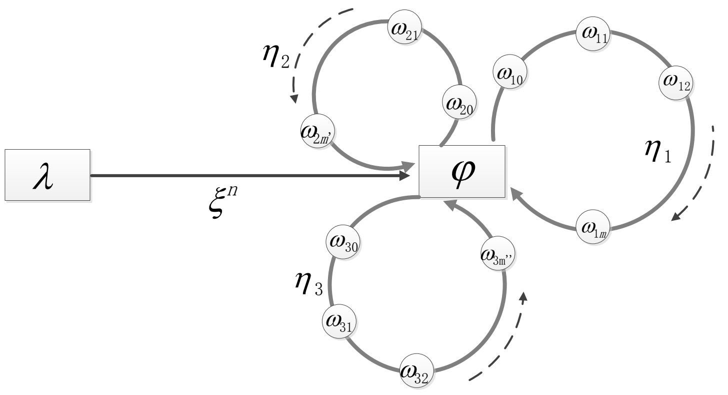

Fig. 3. A snapshot illustrating the relation $\xi \models \lambda$

finite, we can find a looping formula $\varphi = \varphi_i$ that occurs *infinitely often* along this expansion path. On the other side, we can *partition* the run ξ into sequences $\xi = \eta_0\eta_1 \dots$ such each finite sequence η_i is consistent with respect to one loop $\varphi \hookrightarrow \varphi$ along the expansion path. This is illustrated in Figure 3. The definition below formalizes the notion of consistency for finite sequence:

Definition 6. *Let $\eta = \omega_0\omega_1 \dots \omega_n$ ($n \geq 0$) be a finite sequence. Then, we say that η satisfies the LTL formula φ, denoted by $\eta \models_f \varphi$, if the following conditions are satisfied:*

– *there exists $\varphi_0 = \varphi \xrightarrow{\alpha_0} \varphi_1 \xrightarrow{\alpha_1} \dots \xrightarrow{\alpha_n} \varphi_{n+1} = \psi$ such that $\omega_i \models \alpha_i$ for $0 \leq i \leq n$, and with $S := \bigcup_{0\leq j\leq n} CF(\alpha_j)$, it holds*
  1. *if φ is a literal then $\varphi \in S$ holds;*
  2. *if φ is $\varphi_1 U \varphi_2$ or $\varphi_1 R \varphi_2$ then $S \models_f \varphi_2$ holds;*
  3. *if φ is $\varphi_1 \wedge \varphi_2$ then $S \models_f \varphi_1 \wedge S \models_f \varphi_2$ holds;*
  4. *if φ is $\varphi_1 \vee \varphi_2$ then $S \models_f \varphi_1 \vee S \models_f \varphi_2$ holds;*
  5. *if φ is $X\varphi_2$ then $S \models_f \varphi_2$ holds;*

This predicate specifies whether the given finite sequence η is consistent with respect to the finite expansion $\varphi_0 = \varphi \xrightarrow{\alpha_0} \varphi_1 \xrightarrow{\alpha_1} \dots \xrightarrow{\alpha_n} \varphi_{n+1} = \psi$. The condition $\omega_i \models \alpha_i$ requires that the finite sequence η is consistent with respect to the labels along the finite expansion from φ_0. The rules for literals and Boolean connections are intuitive. For Until operator $\varphi_1 U \varphi_2$, it is defined recursively by $S \models_f \varphi_2$: as to make the Until subformula being satisfied, we should make sure that φ_2 holds under S. Similar, for release operator $\varphi_1 R \varphi_2$, we know that $\varphi_1 \wedge \varphi_2$ or φ_2 plays a key role in an accepting run of $\varphi_1 R \varphi_2$. Because $\varphi_1 \wedge \varphi_2$ implies φ_2, and with the rule (4) in the definition, we have $S \models_f \varphi_1 R \varphi_2 \equiv S \models_f \varphi_2$. Assume $\varphi = X\varphi_2$. As $CF(\mathsf{True})$ is defined as $\emptyset$, we have $\eta \models_f \varphi$ iff $\eta' \models_f \varphi_2$ with $\eta' = \omega_1\omega_2 \dots \omega_n$.

The predicate $\models_f$ characterizes whether the prefix of an accepting run contributes to the satisfiability of λ. The idea comes from Corollary 1: Once φ is expanded from itself infinitely by a run ξ as well as $\xi \models \varphi$, there must be some common feature each time

φ loops back to itself. This common feature is what we defined in $\models_f$. In our running example, consider the finite sequence $\eta = \{b, d\}\{b, d\}\{c, e\}$ corresponding to the path $s_1 s_4 s_4 s_1$: according to the definition $\eta \models_f \varphi_1$ holds. For $\eta = \{b, d\}\{b, d\}\{b, d\}$, however, $\eta \not\models_f \varphi_1$.

With the notation $\models_f$, we study below properties for the looping formulas, that will lead to our *central theorem*.

Lemma 4 (Soundness). *Given a looping formula φ and an infinite word ξ, let $\xi = \eta_1\eta_2\ldots$. If $\forall i \geq 1 \cdot \varphi \xrightarrow{\eta_i} \varphi \wedge \eta_i \models_f \varphi$, then $\xi \models \varphi$.*

The soundness property of the looping formula says that if there exists a partitioning $\xi = \eta_1\eta_2...$ such that φ expends to itself by each η_i and $\eta_i \models_f \varphi$ holds, then $\xi \models \varphi$.

Lemma 5 (Completeness). *Given a looping formula φ and an infinite word ξ, if $\varphi \xrightarrow{\xi} \varphi$ and $\xi \models \varphi$ holds, then there exists a partitioning $\eta_1\eta_2\ldots$ for ξ, i.e. $\xi = \eta_1\eta_2\ldots$, such that for all $i \geq 0$, $\varphi \xrightarrow{\eta_i} \varphi \wedge \eta_i \models_f \varphi$ holds.*

The completeness property of the looping formula states the other direction. If $\varphi \xrightarrow{\xi} \varphi$ as well as $\xi \models \varphi$, we can find a partitioning $\eta_1\eta_2\ldots$ that makes φ expending to itself by each η_i and $\eta_i \models_f \varphi$ holds. Combining Lemma 6, Lemma 7 and Corollary 1, we have our central theorem:

Theorem 4 (Central Theorem). *Given a formula λ and an infinite word ξ, we have*

$$\xi \models \lambda \Leftrightarrow \exists \varphi, n \cdot \lambda \xrightarrow{\xi^n} \varphi \wedge \exists \xi_n = \eta_1\eta_2\ldots \cdot \forall i \geq 1 \cdot \varphi \xrightarrow{\eta_i} \varphi \wedge \eta_i \models_f \varphi$$

The central theorem states that given a formula λ, we can always extend it to a looping formula which satisfies the soundness and completeness properties. Reconsider Figure 3: formula λ extends to the looping formula φ by ξ^n, and ξ_n can be partitioned into sequences $\eta_1\eta_2\ldots$. The loops from φ correspond to these finite sequences η_i in the sense $\eta_i \models_f \varphi$.

4.5 Büchi automata for General Formulas

Our central theorem sheds insights about the correspondence between the accepting run and the expansion path from λ. However, how can we guarantee the predicate $\models_f$ for looping formulas in the theorem? We need the last ingredient for starting our automaton construction: we extract the *obligation sets* from LTL formulas that will enable us to characterize $\models_f$.

Definition 7. *Given a formula φ, we define its obligation set, i.e. OS_φ, as follows:*

1. *If $\varphi = p$, $OS_\varphi = \{\{p\}\}$;*
2. *If $\varphi = X\psi$, $OS_\varphi = OS_\psi$;*
3. *If $\varphi = \psi_1 \vee \psi_2$, $OS_\varphi = OS_{\psi_1} \cup OS_{\psi_2}$;*
4. *If $\varphi = \psi_1 \wedge \psi_2$, $OS_\varphi = \{S_1 \cup S_2 \mid S_1 \in OS_{\psi_1} \wedge S_2 \in OS_{\psi_2}\}$;*
5. *If $\varphi = \psi_1 U \psi_2$ or $\psi_1 R \psi_2$, $OS_\varphi = OS_{\psi_2}$;*

For every element set $O \in OS_\varphi$, we call it the obligation of φ.

The obligation set provides all obligations (elements in obligation set) the given formula is supposed to have. Intuitively, a run ξ is accepted a formula φ if ξ can eliminate the obligations of φ. Take the example of $G(aRb)$, the run $(b)^\omega$ is accepted by aRb, and the run eliminates the obligation $\{b\}$ infinitely often.

Notice the similarity of the definition of the obligation set and the predicate $\models_f$. For instance, the obligation set of $\varphi_1 R \varphi_2$ is the obligation set of φ_2, which is similar in the definition of $\models_f$. The interesting rule is the conjunctive one. For obligation set OS_φ, there may be more than one element in OS_φ. However, from the view of satisfiability, if one obligation in OS_φ is satisfied, we can say the obligations of φ is fulfilled. This view leads to the definition of the conjunctive rule. For $\psi_1 \wedge \psi_2$, we need to fulfill the obligations from both ψ_1 and ψ_2, which means we have to trace all possible unions from the elements of OS_{ψ_1} and OS_{ψ_2}. For instance, the obligation set of $G(aUb \wedge cU(d \vee e))$ is $\{\{b, d\}, \{b, e\}\}$. The following lemmas gives the relationship of $\models_f$ and *obligation set*.

Lemma 6. *For all $O \in OS_\varphi$, it holds $O \models_f \varphi$. On the other side, $S \models_f \varphi$ implies that $\exists O \in OS_\varphi \cdot O \subseteq S$.*

For our input formula λ, now we discuss how to construct the Büchi automaton $\mathcal{A}_\lambda$. We first describe the states of the automaton. A state will be consisting of the formula φ and a *process set* that keeps track of properties have been satisfied so far. Formally:

Definition 8 (States of the automaton for λ). *A state is a tuple $\langle \varphi, P \rangle$ where φ is a formula from $EF(\lambda)$, and $P \subseteq AP$ is a process set.*

Refer again to Figure 3: reading the input finite sequence η_1, each element in the process set P_i corresponds to a property set belonging to AP, which will be used to keep track whether all elements in an obligation are met upon returning back to a φ-state. If we have $P_i = \emptyset$, we have successfully returned to the accepting states. Now we have all ingredients for constructing our Büchi automaton $\mathcal{A}_\lambda$:

Definition 9 (Büchi Automaton $\mathcal{A}_\lambda$). *The Büchi automaton for the formula λ is defined as $\mathcal{A}_\lambda = (\Sigma, S, \delta, S_0, \mathcal{F})$, where $\Sigma = 2^{AP}$ and:*

- *$S = \{\langle \varphi, P \rangle \mid \varphi \in EF(\lambda)\}$ is the set of states;*
- *$S_0 = \{\langle \lambda, \emptyset \rangle\}$ is the set of initial states;*
- *$\mathcal{F} = \{\langle \varphi, \emptyset \rangle \mid \varphi \in EF(\lambda)\}$ is the set of accepting states;*
- *Let states s_1, s_2 with $s_1 = \langle \varphi_1, P_1 \rangle$, $s_2 = \langle \varphi_2, P_2 \rangle$ and $w \subseteq 2^{AP}$. Then, $s_2 \in \delta(s_1, \omega)$ iff there exists $\varphi_1 \xrightarrow{\alpha} \varphi_2$ with $\omega \models \alpha$ such that the corresponding P_2 is updated by:*
  1. *$P_2 = \emptyset$ if $\exists O \in OS_{\varphi_2} \cdot O \subseteq P_1 \cup CF(\alpha)$,*
  2. *$P_2 = P_1 \cup CF(\alpha)$ otherwise.*

The transition is determined by the expansion relation $\varphi_1 \xrightarrow{\alpha} \varphi_2$ such that $\omega \models \alpha$. The process set P_2 is updated by $P_1 \cup CF(\alpha)$ unless there is no element set $O \in OS_{\varphi_2}$ such that $P_1 \cup CF(\alpha) \supseteq O$. In that case P_2 will be set to $\emptyset$ and the corresponding state will be recognized as an accepting one.

Now we state the correctness of our construction:

Theorem 5 (Correctness of Automata Generation). *Let λ be the input formula. Then, for any sequence $\xi \in \Sigma^\omega$, it holds $\xi \models \lambda$ iff ξ is accepted by $\mathcal{A}_\lambda$.*

The correctness follows mainly from the fact that our construction strictly adheres to our central theorem (Theorem 4).

We note that two very simple optimizations can be identified for our construction:

- If two states have the same DNF normal form and the same process set P, they are identical. Precisely, we merge states $s_1 = \langle \varphi_1, P_1 \rangle$ and $s_2 = \langle \varphi_2, P_2 \rangle$ if $DNF(\varphi_1) = DNF(\varphi_2)$, and $P_1 = P_2$;
- The elements in the process set P can be restricted into those atomic propositions appearing in OS_φ: Recall here $\varphi \in EF(\lambda)$. One can observe directly that only those properties are used for checking the *obligation* conditions, while others will not be used so that it can be omitted in the process set P.

Now we can finally explain a final detail of our running example:

Example 1. In our running example state s_1 is the accepting state of the automaton. It should be mentioned that the state s_2 = $\langle \varphi_2, \{e\} \rangle$ originally has an edge labeling $c \wedge d$ to the state $\langle \varphi_3, \emptyset \rangle$ according to our construction, which is a new state. However, this state is equivalent with $s_1 = \langle \varphi_1, \emptyset \rangle$, as φ_1 and φ_3 have the same DNF normal form. So these two states are merged. The same cases occur on state s_3 to state s_1 with the edge labeling $b \wedge e$, state s_2 to state s_2 with the edge labeling $b \wedge d$ and etc. After merging these states, we have the automaton as depicted in Figure 2.

Theorem 6 (Complexity). *Let λ be the input formula. Then the Büchi automaton $\mathcal{A}_\lambda$ has the upper bound $2^{2n} + 1$, where n is the number of subformulas in λ.*

Proof. The number of states is bounded by $(2^n + 1) \cdot 2^{|AP|}$. As the first part contains the particular True, the bound is restricted to $2^{2n} + 1$. $\square$

Recall in our construction in the paper we assume all atomic propositions in λ are *syntactically different*, but maintain their *semantically equivalence*. Thus, n referes to the number of subformulas after *tagging* the literals appearing in λ. Details please refer to [8].

5 Experiments

In order to show the efficiency of our construction, we implement the algorithms in our tool *Aalta*[5] and compare it with the SPOT tool [3], which is a state-of-the-art LTL-to-Büchi translator. SPOT follows the tableau framework [15] but uses BDDs [16] to make the translation more efficient. We compare the results between *Aalta* and SPOT's newest 1.0.2 version when this paper is written. Since SPOT has integrated several translations and the results vary on what the user demands so we here choose its default configuration (with flags "-l -t").

[5] http://222.73.57.93/Aalta/

Formula Length	States	Transitions	Nondet-States	Nondet	Time	Product States
10	4.32	17.99	2.69	0.75	0.14	706
	3.44	18.22	1.77	0.74	0.03	538
20	23.30	146.73	4.43	0.82	0.14	4467
	6.67	56.22	2.84	0.76	0.05	1145
30	41.90	259.15	16.32	0.85	0.14	8183
	10.52	113.27	7.62	0.78	0.10	1857
40	45.76	296.05	20.26	0.83	0.06	8909
	20.55	323.20	16.84	0.80	0.27	3857
50	167.13	1161.11	69.52	0.91	0.12	33225
	43.34	744.53	36.87	0.86	2.80	8420

Table 1. Comparison results between SPOT and *Aalta*. In each formula group (with the same length) the first line displays the results from SPOT while the second from *Aalta*.

SPOT also integrates the "randltl"[6] and "ltlcross"[7] scripts which are used to generate random formulas required and check the correctness as well as provide statistics. As a result we can test the correctness of *Aalta* together with obtaining comparing results by using the combination of the two scripts. We set the "randltl" script to generate random formulas with 2 variables with lengths varying from 10 to 50. In the experiments *Aalta* and SPOT are run for around 10,000 formulas, and no errors are detected by the "ltlcross" script. In this paper we assume the results from SPOT are absolutely true, so the testing affirms *Aalta*'s correctness.

In practice, it is not always true that the smaller generated automata are, the better model checking performance is. Etessami et al. [4] pointed out that the model checking performance depends on the size of the product automaton (from the property automaton and the system model) rather than that of property automata. After that Sebastiani and Tonetta [10] conclude the property automata must be "as deterministic as possible". To sum up there are five criteria for evaluating the generated automata: states, transitions, non-deterministic states, non-determinism and the generation time. Among them the transitions are countered deterministically, and the non-deterministic states are those whose outedges are not deterministic. Moreover, since the "ltlcross" script completes simple checking based on the generated automata and a universal model, so we can also gather the size of product states as an important criteria. As a result, we list the comparing results on these criteria in Table 1.

The statistics in Table 1 are arranged as follows. We first set the flags of "randltl" script to fix the alphabet with two variables as well as the probabilities of appearance for temporal operators during formula generation. Then we generate the formulas with length varying from 10 to 50 — 200 formulas for each. The table shows only the average

[6] http://spot.lip6.fr/userdoc/randltl.html

[7] http://spot.lip6.fr/userdoc/ltlcross.html

results in five groups divided by the formula length. Note in each group there are two lines of results in the table, in which the first one corresponds to SPOT and the second corresponds to *Aalta*.

Generally speaking, the smaller the number in the table the better the generated automata are. So one can see that *Aalta* can perform as well as – or even better than – SPOT under the benchmarks we tested. This is very encouraging, as SPOT has been optimized and maintained for more than ten years. The statistics in the table affirm that *Aalta* has better model checking performance than SPOT under some formulas tested. Note the results when the formula length is 50, in which *Aalta* spends more time on automata generation. Thus SPOT has a better scalability, and this is also what need to be improved in *Aalta* in future.

6 Discussion

In this section, we discuss the relationship and differences between our proposed approach and the tableau construction.

Generally speaking, our approach is essentially a tableau one that is based on the expansion laws of Until and Release operators. The interesting aspect of our approach is the finding of a special normal formal with its DNF-based labeled transition system, which is closely related to the Büchi automaton under construction. The tableau approach explicitly expands the formula recursively based on the semantics of LTL formulas while the nodes of the potential automaton are split until no new node can be generated. However, our approach first studies the LTL normal forms to discover the obligations the automaton has to fulfil, and then presents a simple mapping between LTL formulas into Büchi automata.

The insight behind our approach is adopting a different view on the accepting conditions. The tableau approach focuses on the Until-operator. For instance, to decide the accepting states, the tableau approach needs to trace all the Until-subformulas and records the "eventuality" of ψ in $\varphi U \psi$, which leads to the introduction of the *Generalized Büchi Automata* (GBA) in the tableau approach. However, our approach focuses on the *looping formulas*, which potentially consist of the accepting states. Intuitively, an infinite sequence (word) will satisfy the formula λ iff λ can expand to some looping formula φ which can be satisfied by the suffix of the word removing the finite sequence arriving at φ. The key point of our approach is to introduce the static obligation set for each formula in the DNF-based labeled transition system, which indicates that an accepting run is supposed to infinitely fulfil one of the obligations in the obligation set. Thus, the obligation set gives the "invariability" for general formulas instead of the "eventuality" for $Until$-formulas. In the approach, we use a process set to record the obligation that formula φ has been satisfied from its last appearance. Then, we would decide the accepting states easily when the process set fulfills one obligation in the obligation set of φ (We reset it empty afterwards). One can also notice our approach is on-the-fly: the successors of the current state can be obtained as soon as its DNF normal form is acquired.

7 Conclusion

In this paper, we have proposed the *disjunctive-normal forms* for LTL formulas. Based on the DNF representation, we introduce the DNF-based labeled transition system for the input formula λ and study the relationship between the transition system and the Büchi automata for λ. The construction makes use of the notion of obligation set, and we reach a simple but on-the-fly algorithm. When the formula under construction is Release/Until-free, our construction is very straightforward in theory.

Acknowledgement The work is supported by IDEA4CPS and MT-LAB (a VKR Centre of Excellence). The work of Lijun Zhang was done while he was at DTU Compute, Technical University of Denmark, Denmark.

References

1. Tomás Babiak, Mojmír Kretínský, Vojtech Rehák and Jan Strejcek LTL to Büchi Automata Translation: Fast and More Deterministic. In *TACAS*, pages 95–109, 2012.
2. Marco Daniele, Fausto Giunchiglia, and Moshe Y. Vardi. Improved Automata Generation for Linear Temporal Logic. In *CAV*, pages 249–260, 1999.
3. Duret-Lutz, A. and Poitrenaud, D. SPOT: an extensible model checking library using transition-based generalized Büchi automata In *The IEEE Computer Society's 12th Annual International Symposium*, pages 76–83, 2004.
4. Kousha Etessami and Gerard J. Holzmann. Optimizing Büchi Automata. In *CONCUR*, pages 153–167, 2000.
5. Paul Gastin and Denis Oddoux. Fast LTL to Büchi Automata Translation. In *CAV*, pages 53–65, 2001.
6. Rob Gerth, Doron Peled, Moshe Y. Vardi, and Pierre Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *PSTV*, pages 3–18, 1995.
7. Dimitra Giannakopoulou and Flavio Lerda. From States to Transitions: Improving Translation of LTL Formulae to Büchi Automata. In *FORTE*, pages 308–326, 2002.
8. Jianwen Li, Geguang Pu, Lijun Zhang, Zheng Wang, Jifeng He and Kim G. Larsen. On the Relationship between LTL Normal Forms and Buechi Automata. In *CoRR*, 2012.
9. Kristin Y. Rozier and Moshe Y. Vardi. LTL satisfiability checking. In *SPIN*, pages 149–167, 2007.
10. Roberto Sebastiani and Stefano Tonetta. "More Deterministic" vs. "Smaller" Büchi Automata for Efficient LTL Model Checking. In *CHARME*, pages 126–140, 2003.
11. Fabio Somenzi and Roderick Bloem. Efficient Büchi Automata from LTL Formulae. In *CAV*, pages 248–263, 2000.
12. Heikki Tauriainen and Keijo Heljanko. Testing LTL formula translation into Büchi automata. *STTT*, 4(1):57–70, 2002.
13. Moshe Y. Vardi. An Automata-Theoretic Approach to Linear Temporal Logic. In *Banff Higher Order Workshop*, pages 238–266, 1995.
14. Moshe Y. Vardi and Pierre Wolper. An Automata-Theoretic Approach to Automatic Program Verification. In *LICS*, pages 332–344, 1986.
15. Jean-Michel Couvreur. On-the-fly verification of temporal logic. In *FM*, pages 253-271, 1999.
16. Randal E. Bryant Graph-Based Algorithms for Boolean Function Manipulation. In *IEEE Trans. Computers*, pages 677-691, 1986.