Optimal Translation of LTL to Limit Deterministic Automata

Dileep Kini(✉) and Mahesh Viswanathan

Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, USA
kini2@illinois.edu

Abstract. A crucial step in model checking Markov Decision Processes (MDP) is to translate the LTL specification into automata. Efforts have been made in improving deterministic automata construction for LTL but such translations are double exponential in the worst case. For model checking MDPs though limit deterministic automata suffice. Recently it was shown how to translate the fragment LTL$\setminus GU$ to exponential sized limit deterministic automata which speeds up the model checking problem by an exponential factor for that fragment. In this paper we show how to construct limit deterministic automata for full LTL. This translation is not only efficient for LTL$\setminus GU$ but for a larger fragment LTL$_\mathrm{D}$ which is provably more expressive. We show experimental results demonstrating that our construction yields smaller automata when compared to state of the art techniques that translate LTL to deterministic and limit deterministic automata.

1 Introduction

Markov Decision Processes (MDPs) [4,19,23] are the canonical model used to define the semantics of systems like concurrently running probabilistic programs that exhibit both stochastic and nondeterministic behavior. MDPs are interpreted with respect to a scheduler that resolves the nondeterminism. Such a scheduler chooses a probabilistic transition from a state based on the past sequence of states visited during the computation. When undesirable system behaviors are described by a formula φ in linear temporal logic (LTL), *qualitative verification* involves checking if there is some (adversarial) scheduler with respect to which the measure of paths satisfying φ is non-zero. Model checking algorithms [4] in this context proceed by translating the LTL requirement φ into an automaton $\mathcal{A}$, taking the synchronous cross-product of the MDP model M and the automaton $\mathcal{A}$ to construct a new MDP M', and finally, analyzing the MDP M' to check the desired property. The complexity of this procedure is polynomial in the size of the final MDP M', and hence critically depends on the size of automaton $\mathcal{A}$ that results from translating the LTL specification.

D. Kini and M. Viswanathan—Authors were supported by NSF grants CNS 1314485 and CCF 1422798.

A. Legay and T. Margaria (Eds.): TACAS 2017, Part II, LNCS 10206, pp. 113–129, 2017.
DOI: 10.1007/978-3-662-54580-5_7

MDP model checking algorithms based on the above idea require the translated automaton to be of a special form as general non-deterministic automata are not sufficient. The Büchi automaton has to be either deterministic or *deterministic in the limit* — a Büchi automaton is deterministic in the limit if every state reachable from an accepting state has deterministic transitions[1]. Limit-determinism is also sometimes referred to as semi-determinism. Deterministic or limit deterministic automata for LTL formulae can be constructed by first translating the formula into a nondeterministic Büchi automaton, and then either determinizing or "limit-determinizing" the machine. This results in an automaton that is doubly exponential in the size of the LTL formula, which gives a 2EXPTIME algorithm for model checking MDPs.

Direct translations of LTL (and fragments of LTL) to deterministic Rabin automata have been proposed [3,5,10,13,16,17]. However, any such translation, in the worst case, results in automata that are doubly exponential in size [2]; this holds for any fragment of LTL that contains the operators $\vee$, $\wedge$, and $\mathbf{F}$. Recently [8] a fragment of LTL called LTL$\backslash GU$ [14] was translated into limit deterministic Büchi automata. LTL$\backslash GU$ is a fragment of LTL where formulae are built from propositions and their negations using conjunction, disjunction, and the temporal operators $\mathbf{X}$ (next), $\mathbf{F}$ (eventually/finally), $\mathbf{G}$ (always/globally), and $\mathbf{U}$ (until), with the restriction that no $\mathbf{U}$ operator appears within the scope of a $\mathbf{G}$ operator. The most important feature of this translation from LTL$\backslash GU$ to limit deterministic automata is the fact that the resulting automaton is only exponential in the size of the formula. Thus, this automata construction can be used to obtain an EXPTIME algorithm for model checking MDP against LTL$\backslash GU$ formulas, as opposed to 2EXPTIME.

Recently, a translation from full LTL logic to limit deterministic automata has been proposed [20]. This translation is very similar to the translation to deterministic automata proposed in [5], with the use of nondeterminism being limited to simplifying the acceptance condition. Therefore, like the deterministic translations of LTL, it can be shown to construct doubly exponential sized automata even for very simple LTL fragments like those that contain $\vee$, $\wedge$, and $\mathbf{F}$. Thus, it does not achieve the optimal bounds for LTL$\backslash GU$ shown in [8]. However, one advantage of the construction in [20] is that it can be used in quantitative verification as well as qualitative verification of MDPs and has been implemented in [21]. Quantitative verification of MDPs can also be performed using nondeterministic automata that have the *good-for-games* (GFG) property [7,11], but translating a general NBA into a GFG automaton is known to result in an exponential blow-up. An alternate approach to quantitative verification using subset/breakpoint construction on a NBA is proposed in [6] but it also suffers from an exponential blow up.

[1] Limit deterministic automata are not the same as *unambiguous automata*. Unambiguous automata have at most one accepting run for any input. It is well known that every LTL formula can be translated into an unambiguous automaton of exponential size [22]. This has been shown to be not true for limit deterministic automata in [20].

In this paper we continue the line of work started in [8, 20], and present a new translation of the full LTL logic to limit deterministic Büchi automata. The new translation can be shown to be a generalization of the construction in [8] in that it constructs exponential sized automata for LTL$\backslash GU$. In fact, we show that this new translation yields exponential sized automata for a richer fragment of LTL that we call LTL_D (see Sect. 5 for a comparison between the expressive powers of LTL_D and LTL$\backslash GU$). This improves the complexity of qualitative MDP model checking against LTL_D to EXPTIME from 2EXPTIME.

Our automaton construction uses two main ideas. The first is an idea discovered in [8]. To achieve limit determinism, for certain subformulae ψ of φ, the automaton of φ tracks how often $\mathbf{F}\psi$ and $\mathbf{G}\psi$ formulae are true; this is in addition to tracking the truth (implicitly) of all subformulae ψ, as all translations from LTL to automata do. Second, for untils within the scope of $\mathbf{G}$, we do a form of subset construction that ensures that the state explores all the possible ways in which such formulae can be satisfied in the future, and for untils outside the scope of $\mathbf{G}$ we use non-determinism to check its truth.

We have implemented our translation from LTL to limit deterministic automata in a tool called `Büchifier`. We show experimental results demonstrating that in most cases our construction yields smaller automata when compared to state of the art techniques that translate LTL to deterministic and limit deterministic automata.

2 Preliminaries

First we introduce the notation we use throughout the paper. We use P to denote the set of propositions. We use w to denote infinite words over a finite alphabet. We use w_i to denote the i^{th} (index starting at 0) symbol in the sequence w, and use $w[i]$ to denote the suffix $w_i w_{i+1} \dots$ of w starting at i. We use $w[i, j]$ to denote the substring $w_i \dots w_{j-1}$. We use $[n]$ to denote all non-negative integers less than n that is $\{0, 1, \dots, n-1\}$. We begin by recalling the syntax of LTL:

Definition 1 (LTL Syntax). *Formulae in* LTL *are given by the following syntax:*

$$\varphi \quad ::= \quad p \mid \neg p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathbf{X}\varphi \mid \mathbf{F}\varphi \mid \mathbf{G}\varphi \mid \varphi\, \mathbf{U}\, \varphi \qquad\qquad p \in P$$

Next, we look at the semantics of the various operators:

Definition 2 (Semantics). LTL *formulae over a set P are interpreted over words w in $(2^P)^\omega$. The semantics of the logic is given by the following rules*

$$\begin{array}{lcllcl}
w \vDash p\ (\neg p) & \iff & p \in w_0\ (p \notin w_0) & w \vDash \mathbf{X}\varphi & \iff & w[1] \vDash \varphi \\
w \vDash \varphi \vee \psi & \iff & w \vDash \varphi \text{ or } w \vDash \psi & w \vDash \mathbf{F}\varphi & \iff & \exists\, i : w[i] \vDash \varphi \\
w \vDash \varphi \wedge \psi & \iff & w \vDash \varphi \text{ and } w \vDash \psi & w \vDash \mathbf{G}\varphi & \iff & \forall\, i : w[i] \vDash \varphi \\
w \vDash \varphi\, \mathbf{U}\, \psi & \iff & \exists\, i : w[i] \vDash \psi, \text{ and} & & & \\
 & & \forall\, j < i : w[j] \vDash \varphi & & &
\end{array}$$

The semantics of φ, denoted by $[\![\varphi]\!]$, is defined as the set $\{w \in (2^P)^\omega \mid w \vDash \varphi\}$.

(Note that the release operator $\mathbf{R}$, the dual of $\mathbf{U}$, can be expressed using $\mathbf{U}$ and $\mathbf{G}$, i.e. $\psi_1 \, \mathbf{R} \, \psi_2 \equiv (\psi_2 \, \mathbf{U} \, (\psi_1 \wedge \psi_2)) \vee \mathbf{G}\psi_2$. Hence we omit it from any of the logics we consider.)

In this paper the terminology *subformula of* φ is used to denote a node within the parse tree of φ. When we refer to the subformula as an LTL formula we will be referring to the formula at that node. Two subformulae that have the same formulae at their nodes need not be the same owing to the possibility of them being in different contexts. This distinction will be important as we treat formulae differently depending on their contexts. For the purposes of describing different subfragments we qualify subformulae as being either *internal* or *external.*

Definition 3. *A subformula ψ of φ is said to be* internal *if ψ is in the scope of some* $\mathbf{G}$*-subformula of φ, otherwise it is said to be* external.

Many syntactic restrictions of LTL have been considered for the sake of obtaining smaller automata translations. LTL(F,G) (read "LTL F G") and LTL$\backslash GU$ (read "LTL set minus G U") are two such fragments which we recall in the next two definitions.

Definition 4 (LTL(F,G) Syntax). *The fragment* LTL(F,G) *over propositions P is described by the following syntax*

$$\varphi \quad ::= \quad p \mid \neg p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathbf{X}\varphi \mid \mathbf{F}\varphi \mid \mathbf{G}\varphi \qquad\qquad p \in P$$

Definition 5 (LTL\GU Syntax). *The fragment* LTL$\backslash GU$ *is given by the syntax*

$$\psi \quad ::= \quad \varphi \mid \psi \wedge \psi \mid \psi \vee \psi \mid \mathbf{X}\psi \mid \psi \, \mathbf{U} \, \psi \qquad\qquad \varphi \in \text{LTL}(F,G)$$

LTL(F,G) allows for $\mathbf{G}$ and $\mathbf{F}$ as the only temporal operators. The fragment LTL$\backslash GU$ additionally allows for external $\mathbf{U}$ but not internal ones. Also, we choose to represent an external $\mathbf{F}$ using $\mathbf{U}$. In other words every $\mathbf{F}$ will be internal. Next, we introduce the fragment LTL$_\text{D}$ (read "LTL D")

Definition 6 (LTL$_\text{D}$ Syntax). *The formulae in the fragment* LTL$_\text{D}$ *are given by the syntax for ϑ:*

$$\begin{aligned}\psi \quad &::= \quad \varphi \mid \psi \vee \varphi \mid \varphi \vee \psi \mid \psi \wedge \psi \mid \psi \, \mathbf{U} \, \varphi \mid \mathbf{G}\psi \mid \mathbf{X}\psi \qquad\qquad \varphi \in \text{LTL}(F,G)\\ \vartheta \quad &::= \quad \psi \mid \vartheta \vee \vartheta \mid \vartheta \wedge \vartheta \mid \vartheta \, \mathbf{U} \, \vartheta \mid \mathbf{X}\vartheta\end{aligned}$$

Unlike LTL$\backslash GU$, LTL$_\text{D}$ allows for internal $\mathbf{U}$ but it is restricted. The following restrictions apply on LTL$_\text{D}$:

1. The second argument of every internal $\mathbf{U}$ formula is in LTL(F,G)
2. At least one argument of every internal $\vee$ is in LTL(F,G)

Note that LTL$_\text{D}$ is strictly larger than LTL$\backslash GU$ in the syntactic sense, as every LTL$\backslash GU$ formula is also an LTL$_\text{D}$ formula. We shall show in Sect. 5 that it is strictly richer in the semantic sense as well.

Next we define depth and height. A subformula ψ of φ is said to be at depth k if the number of $\mathbf{X}$ operators in φ within which ψ appears is exactly k. The height of a formula is the maximum depth of any of its subformulae.

Definition 7 (Büchi Automata). *A nondeterministic Büchi automaton (NBA) over input alphabet Σ is a tuple (Q, δ, I, F) where Q is a finite set of states; $\delta \subseteq Q \times \Sigma \times Q$ is a set of transitions; $I \subseteq Q$ is a set of initial states and $F \subseteq Q$ is a set of final states.*

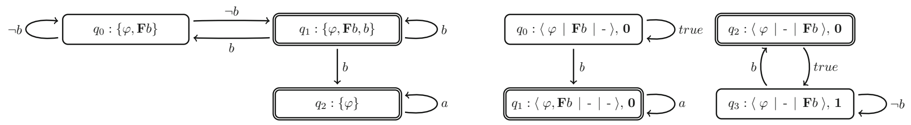
A run of a word $w \in \Sigma^\omega$ over a NBA is an infinite sequence of states $q_0 q_1 q_2 \ldots$ such that $q_0 \in I$ and $\forall\, i \geq 0$ $(q_i, w_i, q_{i+1}) \in \delta$. A run is accepting if $q_i \in F$ for infinitely many i.

The language accepted by an NBA $\mathcal{A}$, denoted by $L(\mathcal{A})$ is the set of all words $w \in \Sigma^\omega$ which have an accepting run on $\mathcal{A}$.

Definition 8 (Limit Determinism). *A NBA (Q, δ, I, F) over input alphabet Σ is said to be* limit deterministic *if for every state q reachable from a final state, it is the case that $|\delta(q, \sigma)| \leq 1$ for every $\sigma \in \Sigma$.*

3 Construction

In this section we show our construction of limit deterministic automata for full LTL. First, let us look at an example that shows that the standard construction (Fischer-Ladner and its variants) is not limit deterministic. The standard construction involves guessing the set of subformulae that are true at each step and ensuring the guess is correct. For $\varphi = \mathbf{G}(a \vee \mathbf{F}b)$ this gives us the automaton (after pruning unreachable states and merging bisimilar ones. Here all 3 states are initial) in Fig. 1a which is not limit deterministic as the final state q_1 has non-deterministic choices enabled.



$\neg b$
$q_0 : \{\varphi, \mathbf{F}b\}$
$\neg b$
b
$q_1 : \{\varphi, \mathbf{F}b, b\}$
b
b
$q_2 : \{\varphi\}$
a
$q_0 : \langle\, \varphi \mid \mathbf{F}b \mid \text{-} \,\rangle, \mathbf{0}$
$true$
$q_2 : \langle\, \varphi \mid \text{-} \mid \mathbf{F}b \,\rangle, \mathbf{0}$
b
b
$true$
$q_1 : \langle\, \varphi, \mathbf{F}b \mid \text{-} \mid \text{-} \,\rangle, \mathbf{0}$
a
$q_3 : \langle\, \varphi \mid \text{-} \mid \mathbf{F}b \,\rangle, \mathbf{1}$
$\neg b$

(a) Standard Construction (b) Tripartition Construction

Fig. 1. Automata for $\mathbf{G}(a \vee \mathbf{F}b)$

Our construction builds upon the idea introduced in [8] of keeping track of how often $\mathbf{F}$,$\mathbf{G}$-subformulae are true. Therefore, we will incrementally describe the features of our automaton: first by revisiting the technique required for LTL(F,G) without $\mathbf{X}$s, later by introducing the new ideas required to handle the untils and nexts.

Given an LTL(F,G) formula, for each of its $\mathbf{G}$-subformula we are going to predict whether it is: always true (α), true at some point but not always (β), never true (γ). Note that for any formula if we predict α/γ then the prediction should remain the same going forward. For a $\mathbf{G}$-subformula, $\mathbf{G}\psi$, if we predict β it means we are asserting $\mathbf{FG}\psi \wedge \neg\mathbf{G}\psi$ and therefore the prediction should remain β until a certain point and then change to α. This prediction entails two kinds of non-deterministic choices: **(i)** the initial choice of assigning one of α, β, γ **(ii)** if assigned β initially then the choice of the time point at which to change it to α. The first choice needs to be made once at the beginning and the second choice has to be made eventually in a finite time. They together only constitute finitely many choices which is the source of the limit determinism. We similarly define predictions for $\mathbf{F}$-subformulae as: never true (α), true at some point but not always (β), always true (γ). We flip the meaning of α and γ to ensure β becomes α eventually as for $\mathbf{G}$-subformulae. An *FG-prediction* for a formula $\varphi \in \text{LTL}(F,G)$, denoted by π, is a tri-partition $\langle \alpha(\pi), \beta(\pi), \gamma(\pi) \rangle$ of its $\mathbf{F}, \mathbf{G}$-subformulae. We drop π when it is clear from the context. The *prediction* for a subformula ψ made by π is said to be $\alpha/\beta/\gamma$ depending upon the partition of π in which ψ is present. The space of all FG-predictions for φ is denoted by $\Pi(\varphi)$.

Example 1. Consider the formula $\varphi = \mathbf{G}(a \vee \mathbf{F}b)$, and an FG-prediction $\pi = \langle \alpha, \beta, \gamma \rangle$ for φ where $\alpha = \{\varphi\}$, $\beta = \{\mathbf{F}b\}$ and $\gamma = \emptyset$. For the formula φ the prediction made is α. Since it is a $\mathbf{G}$-formula this prediction says that φ is always true or simply φ is true. For the subformula $\mathbf{F}b$ the prediction made is β. This prediction says that $\mathbf{F}b$ is true at some point but not always which implies $\mathbf{F}b$ is true but not $\mathbf{GF}b$.

The automaton for $\text{LTL}(F,G)$ essentially makes a non-deterministic choice for π initially and at each step makes a choice of whether to move some formula(e) from β to α. The correctness of predictions made by π is monitored inductively. Suppose our prediction for a formula $\mathbf{G}\psi$ is α at some instant: this implies we need to check that ψ is true at every time point there onwards (or equivalently check that ψ is true whenever α is predicted for $\mathbf{G}\psi$ since the prediction α never changes). If we are able to monitor the ***truth*** of ψ at every instant then it is clear how this can be used to monitor the ***prediction*** α for $\mathbf{G}\psi$. The crucial observation here is that the correct prediction for $\mathbf{G}/\mathbf{F}$ formula gives us their truth: a $\mathbf{G}/\mathbf{F}$ formula is true/false (respectively) at a time point if and only if its correct prediction is α at that time. Now the prediction α for $\mathbf{G}\psi$ can be checked by using the truths (derived from the predictions) of the subformulae of ψ (inductive step). If ψ is propositional then its truth is readily available from the input symbol being seen (base case of the induction). This inductive idea shall be used for all predictions. Note that since our formulae are in negation normal form we only need to verify a prediction is correct if it asserts the truth rather than falsehood of a subformula. Therefore the predictions β, γ for $\mathbf{G}\psi$ need not be checked. In case of $\mathbf{F}\psi$ the prediction α need not be checked (as it entails falsehood of $\mathbf{F}\psi$) but β, γ do need to be checked. If our prediction for $\mathbf{F}\psi$ is β then we are asserting ψ is true until a certain point in the future at which the prediction becomes α. Therefore we only need to check that ψ is true when the prediction for $\mathbf{F}\psi$ changes to α. Once again we can inductively obtain the truth of ψ at that instant from the predictions for the subformulae of ψ and from the next input. For checking a prediction γ about $\mathbf{F}\psi$ we need to check ψ is true infinitely often. For this purpose we use the Büchi acceptance where the final states are those where ψ is evaluated to be true, again inductively. When we are monitoring multiple $\mathbf{F}\psi$ for γ we will need a counter to cycle through all the $\mathbf{F}\psi$ in γ. Let m be the number of $\mathbf{F}\psi$ in γ. Observe that the set of formulae predicted to be γ never changes once fixed at the beginning and hence m is well defined. When the counter has value n, it is incremented cyclically to $n + 1 (\text{mod } m)$ whenever the ψ corresponding to the n^{th} $\mathbf{F}\psi \in \gamma$ evaluates to true. The initial states are those in which the top formula evaluates to true given the predictions in that state. The final states are those where no formula is assigned β and the counter is 0. Summarizing, a state in our automata has two components: **(a)** an FG-prediction $\pi = \langle \alpha, \beta, \gamma \rangle$ (a tri-partition of the $\mathbf{F}, \mathbf{G}$-subformulae) and **(b)** a cyclic integer counter n. The transitions are determined by how the predictions and counters are allowed to change as described. We illustrate the construction using once again the formula $\varphi = \mathbf{G}(a \vee \mathbf{F}b)$ for which the automaton is presented in Fig. 1b and its details are completely described in the technical report [9].

3.1 Handling Untils and Nexts

Next we observe that the above technique does not lend itself to the $\mathbf{U}/\mathbf{X}$ operators. The crucial property used above about $\mathbf{F}, \mathbf{G}$-formulae is that they cannot be simultaneously infinitely often true and infinitely often false unlike $\mathbf{U}/\mathbf{X}$ formulae. So if we tried the above technique for $\mathbf{U}/\mathbf{X}$ we would not get limit determinism since the truth of the $\mathbf{U}/\mathbf{X}$ formulae would have to be guessed infinitely often.

The key idea we use in handling $\mathbf{U}/\mathbf{X}$s is to propagate their *obligation* along the states. Let us say the automaton needs to check if a formula φ holds for an input w, and it begins by making an FG-prediction π about w. The obligation when no input has been seen is φ. When the first symbol w_0 is seen it needs to update the obligation to reflect what "remains to be checked" for the rest of the input $w[1]$, in order for $w \vDash \varphi$ to hold, assuming π is correct for w. The automaton can keep updating the obligation as it sees each input symbol. The claim will be that the obligation is never falsified iff $w \vDash \varphi$, given that π is correct. This brings up some questions:

1. How are we exploiting opportunities for non-determinism?
2. How is the obligation computed at each step?
3. How is π checked to be correct in the presence of $\mathbf{U}/\mathbf{X}$s?

Exploiting Non-determinism. Being able to exploit non-determinism helps in reducing the size of the automaton we construct. So the question is: how are we exploiting any opportunities for non-determinism (albeit for finite time)? The answer is to update the obligation non-deterministically. Checking the formula $\psi_1 \mathbf{U} \psi_2$ presents us with two alternatives: either ψ_2 is true now or $\psi_1 \wedge \mathbf{X}(\psi_1 \mathbf{U} \psi_2)$ is true now. Similarly $\psi_1 \vee \psi_2$ brings up two alternatives. We can pick between the obligations of these two choices non-deterministically. But we should somehow

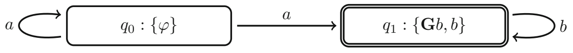


Fig. 2. Standard NBA construction for $\varphi = a\mathbf{U}(\mathbf{G}b)$.

ensure that we are only allowed to use this non-determinism finitely often. This is where we treat internal and external (Definition 3) $\mathbf{U}/\vee$ subformulae differently. The observation is that external $\mathbf{U}/\vee$ need to be checked for only a finite amount of time. Hence the disjunctive choice presented by them can be dispatched non-deterministically each time without worrying about violating limit determinism. To illustrate this point we show the standard NBA for the formula $\varphi = a\mathbf{U}(\mathbf{G}b)$ in Fig. 2 which turns out to be limit deterministic owing to the fact that the $\mathbf{U}$ is external. In Fig. 1a we saw that the standard construction for $\varphi = \mathbf{G}(a \vee \mathbf{F}b)$ resulted in a NBA that was not limit-deterministic, and one of the reasons is that the $\mathbf{F}$, which is a special form of $\mathbf{U}$, is internal. An internal $\mathbf{U}/\vee$ may need to be checked infinitely many times and hence the choice should not be resolved non-deterministically, but carried forward as a disjunction of the obligations of the choices. Passing the choice forward without resolving it comes at a cost of a bigger state space, this is akin to the subset construction where all the choices are being kept track of.

Now we begin to formalize the ideas. To exploit the non-determinism allowed by the external $\mathbf{U}/\vee$ we introduce the concept of *ex-choice*. We use Λ_φ to denote the set of all external $\mathbf{U}/\vee$ subformulae. Any subset of it $\lambda \subseteq \Lambda_\varphi$ is called an ex-choice. An ex-choice dictates how each external $\mathbf{U}/\vee$ should be satisfied if it needs to be satisfied. The interpretation associated with λ is the following: if $\psi_1\mathbf{U}\psi_2 \in \lambda$ then ψ_2 has to hold or if $\psi_1\mathbf{U}\psi_2 \in \Lambda_\varphi - \lambda$ then $\psi_1 \wedge \mathbf{X}(\psi_1\mathbf{U}\psi_2)$ has to hold. Similarly if $\psi_1 \vee \psi_2 \in \lambda$ then ψ_1 has to hold and if $\psi_1 \vee \psi_2 \in \Lambda_\varphi - \lambda$ then ψ_2 has to hold. The automaton we are going to construct is going to non-deterministically pick an ex-choice at each step and use it resolve the choices on external $\mathbf{U}/\vee$. After a finite time the ex-choice will not matter as the obligations will not consist of any external $\mathbf{U}/\vee$ that need to be checked (which will be enforced as a part of the acceptance condition), and hence limit determinism is ensured. The ex-choice picked along a transition is going to determine the obligation computed. Which leads us to the question of how the obligation is computed.

Computing Obligation. We define the *derivative* of a formula μ w.r.t an input symbol σ, FG-prediction π and ex-choice λ. The derivative should capture the obligation/requirement on any word ρ such that those obligations are able to imply that $\sigma\rho$ satisfies μ. This enables us to keep passing on the obligation forward as we see each symbol of the input by taking the derivative of the obligation so far. First, we need to ensure that the ex-choice λ picked when we are taking the transition dictates how a formula in Λ_φ should be satisfied if it needs to be. With that in mind we define $f(\lambda)$ as follows:

$$\begin{aligned} f(\lambda) = {} & (\wedge_{(\phi\mathbf{U}\psi \in \lambda)} \phi \,\mathbf{U}\, \psi \Rightarrow \psi) \wedge (\wedge_{(\phi\mathbf{U}\psi \in (\Lambda_\varphi - \lambda))} \phi \,\mathbf{U}\, \psi \Rightarrow (\phi \wedge \mathbf{X}(\phi \,\mathbf{U}\, \psi))) \\ & \wedge (\wedge_{(\phi \vee \psi \in \lambda)} \phi \vee \psi \Rightarrow \phi) \wedge (\wedge_{(\phi \vee \psi \in \Lambda_\varphi - \lambda)} \phi \vee \psi \Rightarrow \psi) \end{aligned}$$

Since predictions made by π already tell us the truth of some of the subformulae, they need to be taken into account. Towards that we define the *substitution* of a formula ϕ w.r.t π, denoted by $[\,\phi\,]_\pi$ as the formula obtained from ϕ by substituting occurrences $\mathbf{G}\psi$ with $\mathbf{tt}$ if $\mathbf{G}\psi \in \alpha$ and $\mathbf{ff}$ otherwise, and similarly for $\mathbf{F}\psi$ with $\mathbf{ff}$ if $\mathbf{F}\psi \in \alpha$ and $\mathbf{tt}$ otherwise. The substitutions are done only for the maximal formulae in π that appear in ϕ, i.e., if ψ_1, ψ_2 are formulae in π such that ψ_1 is a subformula of ψ_2 then the substitution is not performed for ψ_1. Now we are ready to give a declarative definition of the derivative:

Definition 9. *Given an LTL formula μ over P, and a triple $\varepsilon = (\sigma, \pi, \lambda)$ where $\sigma \in 2^P$, $\pi \in \Pi(\varphi)$ and $\lambda \subseteq \Lambda_\varphi$: an LTL formula ψ is said to be a* ***derivative*** *of μ w.r.t to ε if*

$$\forall \rho \in \left(2^P\right)^\omega \quad \rho \vDash \psi \implies \sigma\rho \vDash [\,\mu \wedge f(\lambda)\,]_\pi$$

The ***weakest derivative*** *of μ w.r.t ε, denoted by $\nabla(\mu, \varepsilon)$, is a derivative such that $\psi \implies \nabla(\mu, \varepsilon)$ for any other derivative ψ.*

Since we will only be interested in the weakest derivative (as opposed to any other derivative) we shall refer to it as the derivative. The above definition is only declarative in the sense that it does not give us an explicit way to compute the derivative. We present this definition here for the sake of simplicity and ease of understanding for the reader. In the companion technical report [9] we provide a syntactic definition and all the necessary machinery that allows us to compute such a formula. The syntactic definition also restricts the representation of the obligations to $\mathcal{B}^+(\varphi)$ which is the set of all positive Boolean combinations of subformulae of φ.

The automaton now will have an extra component μ corresponding to the obligation along with (π, n) from before. In the initial state μ will be the given formula φ that needs to be checked. At each step, the automaton sees an input symbol σ and makes a non-deterministic ex-choice $\lambda \subseteq \Lambda_\varphi$. The obligation at the next state will then become $\nabla(\mu, \varepsilon)$ where $\varepsilon = (\sigma, \pi, \lambda)$. The process continues as long as the obligation is never falsified. In order to ensure that every external until is dispatched in finite time, we impose that the obligation μ in the final states is *ex-free*, i.e. free of any formulae in Λ_φ. When the obligation is ex-free the ex-choice does not play a role in determining its derivative and we shall drop λ whenever that is the case, and this eliminates any non-determinism once a final state is visited. In order to ensure that an internal until, say $\phi \,\mathbf{U}\, \psi$ is not delayed forever, we involve $\mathbf{F}\psi$ in the FG-prediction and enhance the definition of substitution to say that $\phi \,\mathbf{U}\, \psi$ is replaced with $\mathbf{ff}$ if $\mathbf{F}\psi \in \alpha$. This way the derivative will impose that $\mathbf{F}\psi$ is true whenever $\phi \,\mathbf{U}\, \psi$ is claimed to be true. With this in mind we define the closure of φ, denoted by $\mathcal{C}(\varphi)$, to be set of all $\mathbf{F}, \mathbf{G}$-subformulae of φ, along with all $\mathbf{F}\psi$ for every internal $\phi\mathbf{U}\psi$ subformula of φ. We re-define an FG-prediction π to be any tri-partition of $\mathcal{C}(\varphi)$. Note that for every $\mathbf{F}\psi$ or $\mathbf{G}\psi$ in $\mathcal{C}(\varphi)$, ψ is internal.

Example 2. Let $\varphi = \mathbf{G}(\mathbf{F}a \vee (b \,\mathbf{U}\, c))$. Here $\mathcal{C}(\varphi) = \{\varphi, \mathbf{F}a, \mathbf{F}c\}$.

Example 3. Let $\varphi = a\mathbf{U}(b \wedge \mathbf{G}c)$ be an internal subformula of some given formula. $\nabla(\varphi, \varepsilon)$ can take different values depending upon $\varepsilon = (\sigma, \pi)$. Here ex-choice λ does not play a role because the only $\mathbf{U}$ is internal. Note that $\varphi' = \mathbf{F}(b \wedge \mathbf{G}c)$ is in the closure. If $\varphi' \in \alpha$, then $\nabla(\varphi, \varepsilon) = \mathbf{ff}$ because $[\varphi]_\pi$ would be $\mathbf{ff}$ owing to φ being substituted with $\mathbf{ff}$. Let $\varphi' \notin \alpha$. Now if $\mathbf{G}c \in \alpha$ then substituting $\mathbf{tt}$ in place of $\mathbf{G}c$ gives us $a\mathbf{U}b$ whose satisfaction depends upon the truth of a and b as given by σ. So if $\sigma(b) = \mathbf{tt}$ then the $\mathbf{U}$ is immediately satisfied and so $\nabla(\varphi, \varepsilon) = \mathbf{tt}$. If $\sigma(b) = \mathbf{ff}$ then the $\mathbf{U}$ is delayed and hence $\nabla(\varphi, \varepsilon)$ is either $a\mathbf{U}b$ or $\mathbf{ff}$ depending on $\sigma(a) = \mathbf{tt}/\mathbf{ff}$ respectively. If $\mathbf{G}c \notin \alpha$ then truth of b does not matter (as replacing $\mathbf{G}c$ with $\mathbf{ff}$ makes $b \wedge \mathbf{G}c = \mathbf{ff}$) and once again the derivative is $\varphi/\mathbf{ff}$ depending upon $\sigma(a)$.

Checking FG-Predictions in the Presence of Untils and Nexts. The main idea in being able to check an FG-prediction π was that a correct prediction about an $\mathbf{F}, \mathbf{G}$-subformula also tells us its truth. When we have $\mathbf{U}/\mathbf{X}$s in the mix, we no longer have a prediction available for them, and hence no immediate way to check if some subformula is true. For example when $\mathbf{G}\psi \in \alpha$ we needed to check ψ is true and we did so inductively using the predictions for subformulae in ψ. Now, since ψ can have $\mathbf{U}/\mathbf{X}$ within them it is not clear how we are going to check truth of ψ. In this case we pass ψ to the obligation μ. Similarly when the prediction of $\mathbf{F}\psi$ is changed from β to α we need to check ψ is true so once again we pass ψ to the obligation. So given consecutive FG-predictions π, π' define Ψ as the set

$$\Psi = \{\psi \mid \mathbf{F}\psi \in \beta(\pi) \cap \alpha(\pi') \text{ or } \mathbf{G}\psi \in \alpha(\pi)\} \tag{1}$$

and update the obligation along a transition $(\mu, \pi, n) \xrightarrow{\sigma} (\mu', \pi', n')$ as: $\mu' = \nabla(\mu \wedge (\wedge_{\psi \in \Psi} \psi), \varepsilon)$ where $\varepsilon = (\sigma, \pi, \lambda)$. Now consider the case when the counter is $n > 0$ and need to verify that the nth $\mathbf{F}\psi$ formula in γ is true. In this case we cannot pass on ψ to the obligation because $\mathbf{F}\psi$ may be true because ψ is true at a later point and not now. Since we cannot predict when ψ is going to be true we carry the disjunction of all the derivatives of ψ since the counter was incremented to n. We keep doing it until this "carry" becomes true indicating that ψ became true at some point since we started checking for it. We also increment the counter at that point. This "carry" becomes yet another component ν in the automaton's state. We use $\mathbb{F}(S)$ to denote all $\mathbf{F}\psi$ in set S. Now we are ready to put the pieces together to formally describe the entire construction.

Definition 10 (Construction). *Given a formula $\varphi \in$ LTL over propositions P, let $\mathcal{D}(\varphi)$ be the NBA (Q, δ, I, F) over the alphabet 2^P defined as follows:*

- *Q is the set $\mathcal{B}^+(\varphi) \times \mathcal{B}^+(\varphi) \times \Pi(\varphi) \times [n]$ where $n = |\mathbb{F}(\mathcal{C}(\varphi))| + 1$*
- *δ is the set of all transitions $(\mu, \nu, \pi, m) \xrightarrow{\sigma} (\mu', \nu', \pi', m')$ such that*
  - **(a)** *$\alpha(\pi) \subseteq \alpha(\pi')$ and $\gamma(\pi) = \gamma(\pi')$*
  - **(b)** *$\mu' = \nabla(\mu \wedge \theta, \varepsilon)$ for some $\lambda \subseteq \Lambda_\varphi$ where $\theta = (\wedge_{\psi \in \Psi} \psi)$, Ψ as defined in (1) and $\varepsilon = (\sigma, \pi, \lambda)$*

(c) $m' = \begin{cases} (m+1)\,(mod\,|\mathbb{F}(\gamma)|+1) & \nu = \mathbf{tt} \\ m & \text{otherwise} \end{cases}$

(d) $\nu' = \begin{cases} \psi_{m'} & \nu = \mathbf{tt} \\ \nabla(\nu, \varepsilon) \vee \psi_m & \text{otherwise} \end{cases}$

where $\{\mathbf{F}\psi_1, .., \mathbf{F}\psi_k\}$ *is an enumeration of* $\mathbb{F}(\gamma)$*,* $\psi_0 = \mathbf{tt}$ *and* $\varepsilon = (\sigma, \pi)$

- *I is all states of the form $(\varphi, \mathbf{tt}, \pi, 0)$*
- *F is all states of the form $(\mu, \mathbf{tt}, \pi, 0)$ where $\beta(\pi) = \emptyset$, $\mu \neq \mathbf{ff}$, μ is ex-free*

We state the correctness result here and include the proofs in the technical report [9].

Theorem 1. *For $\varphi \in$ LTL, $\mathcal{D}(\varphi)$ is a limit deterministic automaton such that $L(\mathcal{D}(\varphi)) = [\![\varphi]\!]$ and $\mathcal{D}(\varphi)$ is of size at most double exponential in φ.*

The number of different formulae in $\mathcal{B}^+(\varphi)$, is at most double exponential in the size of φ, since each can be represented as a collection of subsets of subformulae of φ. $\Pi(\varphi)$ is simply tripartition of $\mathcal{C}(\varphi)$ which is bounded above by $3^{|\varphi|}$. And the counter can take $|\mathbb{F}(\mathcal{C}(\varphi))| + 1$ different values which is $\leq |\varphi|$. The entire state space $\mathcal{B}^+(\varphi) \times \mathcal{B}^+(\varphi) \times \Pi(\varphi) \times [n]$ is upper bounded by the product of these which is clearly doubly exponential.

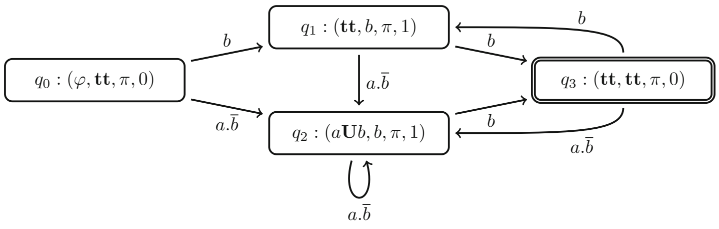

$q_0 : (\varphi, \mathbf{tt}, \pi, 0)$
$q_1 : (\mathbf{tt}, b, \pi, 1)$
$q_2 : (a\mathbf{U}b, b, \pi, 1)$
$q_3 : (\mathbf{tt}, \mathbf{tt}, \pi, 0)$
b
$a.\bar{b}$

Fig. 3. Our construction for $\varphi = \mathbf{G}(a\mathbf{U}b)$.

We illustrate our construction using $\varphi = \mathbf{G}(a\mathbf{U}b)$ which is a formula outside LTL$\backslash GU$. The automaton for φ is shown in Fig. 3. First note that the $\mathcal{C}(\varphi) = \{\varphi, \mathbf{F}b\}$. Next, observe that the only interesting FG-prediction is π in which $\alpha = \{\varphi\}$, $\beta = \emptyset$ and $\gamma = \{\mathbf{F}b\}$. This is because any initial state will have $\mu = \varphi$ which forces $\varphi \in \alpha$, and since predictions in α don't change, every reachable state will have $\varphi \in \alpha$ as well. As for $\mathbf{F}b$ note that the corresponding internal until $a\mathbf{U}b$ will become $\mathbf{ff}$ if $\mathbf{F}b$ is in α and thus making the derivative $\mathbf{ff}$ ($a\mathbf{U}b$ is added to the obligation at each step since $\varphi \in \alpha$ and rule **(b)**). Therefore $\mathbf{F}b$ cannot be in α, and it cannot be in β because then it would be eventually in α. So $\mathbf{F}b$ has to be in γ. Now that π is fixed, and given input σ, the obligation μ changes according to rule **(b)** as $\mu' = \nabla(\mu \wedge (a\mathbf{U}b), (\sigma, \pi))$. Similarly the carry ν changes to b if $\nu = \mathbf{tt}$ (as in q_3 to q_1/q_2) and becomes $\nu' = \nabla(\nu, (\sigma, \pi)) \vee b$

otherwise in accordance with rule **(d)**. The initial state is q_0 with $\mu = \varphi$, $\nu = \mathbf{tt}$ and counter $= 0$. The counter is incremented whenever ν becomes $\mathbf{tt}$. It is easy to see that the automaton indeed accepts $\mathbf{G}(a\,\mathbf{U}\,b)$ and is limit deterministic.

4 Efficiency

In this section we state the results regarding the efficiency of our construction for LTL_D. We prove that there are only exponentially many reachable states in $\mathcal{D}(\varphi)$. A state $q = (\mu, \nu, \pi, n)$ of $\mathcal{D}(\varphi)$ is called reachable if there exists a valid finite run of the automaton that ends in q. A μ is said to be reachable if (μ, ν, π, n) is reachable for some choice of ν, π and n. Similarly for ν. We show that the space of reachable μ and ν is only exponentially large in the size of φ. Our approach will be to show that every reachable μ (or ν) can be expressed in a certain way, and we will count the number of different such expressions to obtain an upper bound. The expression for μ and ν relies on them being represented in DNF form and uses the syntactic definition of the derivative given in the technical report [9]. Therefore we state only the main result and its consequence on the model checking complexity here and present the proofs in [9].

Theorem 2. *For $\varphi \in \mathrm{LTL}_\mathrm{D}$ the number of reachable states in the $\mathcal{D}(\varphi)$ is at most exponential in $|\varphi|$.*

Theorem 3. *The model checking problem for MDPs against specification in* LTL_D *is* EXPTIME-*complete*

Proof. The upper bound follows from our construction being of exponential size and the fact that the model checking of MDPs can be done by performing a linear time analysis of the synchronous product of the MDP and the automaton [4]. The EXPTIME hardness lower bound is from the fact that the problem is EXPTIME hard for the subfragment $\mathrm{LTL}\backslash GU$ as proved in [8].

5 Expressive Power of LTL_D

In this section we show that LTL_D is semantically more expressive than $\mathrm{LTL}\backslash GU$. We demonstrate that the formula $\varphi_0 = \mathbf{G}(p \vee (q\,\mathbf{U}\,r))$ which is expressible in LTL_D, cannot be expressed by any formula in $\mathrm{LTL}\backslash GU$.

Let us fix integers $\ell, k \in \mathbb{N}$. We will use $\mathrm{LTL}_\ell(F, G)$ to denote the subfragment of $\mathrm{LTL}(F,G)$ where formulae have maximum height ℓ. Since $\mathbf{X}$ distributes over all other operators we assume that all the $\mathbf{X}$s are pushed inside. We use $\mathrm{LTL}_{\ell,k}\backslash GU$ to denote the fragment where formulae are built out of $\mathbf{U}$, $\wedge$, $\vee$ and $\mathrm{LTL}_\ell(F, G)$ formulae such that the number of $\mathbf{U}$s used is at most k.

Next, consider the following strings over 2^P where $P = \{p, q, r\}$:

$$u = \{p\}\{p,q\}^\ell\{p\} \quad v = \{q\}\{p,q\}^\ell\{r\} \quad w = \{q\}\{p,q\}^\ell\{p\}$$
$$s_k = (uv)^{k+1}u \qquad \sigma = (uv)^\omega \qquad \eta_k = s_k w v \sigma$$

The observation we make is that σ satisfies φ_0 but η_k does not. We state the main theorem and the corollary here and leave the details in the tech report [9].

Theorem 4. $\forall \varphi \in \mathrm{LTL}_{\ell,k} \backslash GU \quad \sigma \vDash \varphi \implies \eta_k \vDash \varphi$

Corollary 1. *φ_0 is not expressible in $\mathrm{LTL}_{\ell,k} \backslash GU$. Also since ℓ and k are arbitrary, φ_0 is not expressible in $\mathrm{LTL} \backslash GU$.*

6 Experimental Results

We present our tool `Büchifier` (available at [1]) that implements the techniques described in this paper. `Büchifier` is the first tool to generate LDBA with provable exponential upper bounds for a large class of LTL formulae. The states (μ, ν, π, n) in our automaton described in Definition 10, involve $\mu, \nu \in \mathcal{B}^+(\varphi)$ which are essentially sets of sets of subformulae. We view each subformula as a different proposition. We then interpret the formulae in $\mathcal{B}^+(\varphi)$ as a Boolean function on these propositions. In `Büchifier` we represent these Boolean functions symbolically using Binary Decision Diagrams (BDD). Our overall construction follows a standard approach where we begin with an initial set of states and keep adding successors to discover the entire reachable set of states. We report the number of states, number of transitions and the number of final states for the limit deterministic automata we construct.

MDP model checkers like PRISM [15], for a long time have used the translation from LTL to deterministic Rabin automata and only recently [20] have started using limit deterministic Büchi automata. As a consequence we compare the performance of our method against `Rabinizer 3` [12] (the best known tool for translating LTL to deterministic automata) and `ltl2ldba` [20] (the only other known tool for translating LTL to LDBA). `Rabinizer 3` constructs deterministic Rabin automata with generalized Rabin pairs (DGRA). The experimental results in [5,12] report the size of DGRA using the number of states and number of acceptance pairs of the automata; the size of each Rabin pair is, unfortunately, not reported. Since the size of Rabin pairs influences the efficiency of MDP model checking, we report it here to make a meaningful comparison. We take the size of a Rabin pair to be simply the number of transitions in it. The tool `ltl2ldba` generates transition-based generalized Büchi automata (TGBA). The experimental results in [20] report the size of the TGBA using number of states and number of acceptance sets, and once again the size of each of these sets is not reported. Since their sizes also effect the model checking procedure we report them here. We take the size of an acceptance set to be simply the number of transitions in it. In Table 1 we report a head to head to comparison of `Büchifier`, `Rabinizer 3` and `ltl2ldba` on various LTL formulae.

1. The first 5 formulae are those considered in [5]; they are from the GR(1) fragment [18] of LTL. These formulae capture Boolean combination of fairness conditions for which generalized Rabin acceptance is particularly well suited. `Rabinizer 3` does well on these examples, but `Büchifier` is not far behind its competitors. The formulae are instantiations of the following templates: $g_0(j) = \wedge_{i=1}^{j}(\mathbf{GF}a_i \Rightarrow \mathbf{GF}b_i)$, $g_1(j) = \wedge_{i=1}^{j}(\mathbf{GF}a_i \Rightarrow \mathbf{GF}a_{i+1})$.

Table 1. A Comparison between the sizes of automata produced by `Büchifier`, `Rabinizer 3` and `ltl2ldba` on various formulae. Column St denotes the number of states, column Tr denotes the number of transitions and column AC denotes the size of the acceptance condition. Entries marked as "–" indicate that the tool failed to construct the automaton and/or the acceptance condition due to the memory limit (1 GB) being exceeded.

	`Büchifier`			`Rabinizer 3`			`ltl2ldba`		
	St	Tr	AC	St	Tr	AC	St	Tr	AC
$g_0(1)$	4	7	2	1	1	3	3	6	2 (1)
$g_0(2)$	12	23	5	1	1	8	5	14	12 (2)
$g_0(3)$	32	63	8	1	1	20	9	36	54 (3)
$g_1(2)$	12	21	5	1	1	8	5	13	11 (2)
$g_1(3)$	31	54	13	1	1	18	9	30	44 (3)
φ_1	5	7	3	5	13	40	7	23	12 (4)
φ_2	26	83	8	12	48	233	36	101	75 (2)
φ_3	13	25	3	16	128	64	21	140	129 (2)
φ_4	17	47	7	2	4	35	9	29	31 (2)
φ_5	36	111	11	12	48	330	41	133	94 (2)
$f_0(1)$	4	7	2	2	4	2	2	4	2 (1)
$f_0(2)$	14	29	5	16	74	26	4	16	16 (2)
$f_0(3)$	44	105	13	–	–	–	8	64	96 (3)
$f_0(4)$	130	369	33	–	–	–	16	256	512 (4)
$f_1(1)$	14	29	5	6	24	10	8	32	12 (1)
$f_1(2)$	130	369	33	–	–	–	64	1024	768 (2)
$f_1(3)$	1050	4801	193	–	–	–	512	32768	36K (3)
$f_2(1)$	1	1	1	2	3	2	1	1	2 (2)
$f_2(2)$	5	7	3	5	13	45	6	21	9 (3)
$f_2(3)$	19	37	7	19	109	847	19	218	28 (4)
$f_2(4)$	65	175	15	167	2529	–	93	6301	75 (5)
$f_3(1)$	2	4	1	3	7	4	1	2	3 (2)
$f_3(2)$	10	20	4	17	91	53	14	62	28 (1)
$f_3(3)$	36	78	12	–	–	–	212	2359	953 (1)
$f_3(4)$	114	288	32	–	–	–	17352	598330	167K (1)
$h(2,1)$	26	54	9	15	49	49	14	44	1 (1)
$h(2,2)$	60	138	21	65	469	469	64	434	1 (1)
$h(2,3)$	182	468	57	315	5119	5119	314	4892	1 (1)
$h(4,1)$	80	146	36	76	250	250	75	229	1 (1)
$h(4,2)$	230	464	96	990	8068	8068	989	7465	1 (1)
$h(4,3)$	908	1994	348	–	–	–	–	–	–
ψ_1	35	62	9	3	6	12	3	6	8 (3)
ψ_2	7	15	3	8	39	53	2	5	18 (3)
ψ_3	29	62	8	29	116	74	62	293	27 (2)
ψ_4	26	92	6	4	11	7	3	8	3 (1)
ψ_5	9	58	1	5	17	9	3	9	3 (1)

2. The next 5 formulae are also from [5] to show how `Rabinizer 3` can effectively handle $\mathbf{X}$s. `Büchifier` has a comparable number of states and much smaller acceptance condition when compared to `Rabinizer 3` and `ltl2ldba` in all these cases. $\varphi_1 = \mathbf{G}(q \vee \mathbf{XG}p) \wedge \mathbf{G}(r \vee \mathbf{XG}\neg p)$, $\varphi_2 = (\mathbf{GF}(a \wedge \mathbf{X}^2 b) \vee \mathbf{FG}b) \wedge \mathbf{FG}(c \vee (\mathbf{X}a \wedge \mathbf{X}^2 b))$, $\varphi_3 = \mathbf{GF}(\mathbf{X}^3 a \wedge \mathbf{X}^4 b) \wedge \mathbf{GF}(b \vee \mathbf{X}c) \wedge \mathbf{GF}(c \wedge \mathbf{X}^2 a)$, $\varphi_4 = (\mathbf{GF}a \vee \mathbf{FG}b) \wedge (\mathbf{GF}c \vee \mathbf{FG}(d \vee \mathbf{X}e))$, $\varphi_5 = (\mathbf{GF}(a \wedge \mathbf{X}^2 c) \vee \mathbf{FG}b) \wedge (\mathbf{GF}c \vee \mathbf{FG}(d \vee (\mathbf{X}a \wedge \mathbf{X}^2 b)))$.
3. The next 15 formulae (4 groups) express a variety of natural properties, such as $\mathbf{G}(req \Rightarrow \mathbf{F}ack)$ which says that every request that is received is eventually acknowledged. As shown in the table in many of the cases `Rabinizer 3` runs out of memory (1 GB) and fails to produce an automaton, and `ltl2ldba` fails to scale in comparison with `Büchifier`. The formulae in the table are instantiations of the following templates: $f_0(j) = \mathbf{G}(\wedge_{i=1}^{j}(a_i \Rightarrow \mathbf{F}b_i))$, $f_1(j) = \mathbf{G}(\wedge_{i=1}^{j} (a_i \Rightarrow (\mathbf{F}b_i \wedge \mathbf{F}c_i)))$, $f_2(j) = \mathbf{G}(\vee_{i=1}^{j}(a_i \wedge \mathbf{G}b_i))$, $f_3(j) = \mathbf{G}(\vee_{i=1}^{j}(a_i \wedge \mathbf{F}b_i))$.
4. The next 6 formulae expressible in LTL$\backslash GU$, contain multiple $\mathbf{X}$s and external $\mathbf{U}$s. `Büchifier` constructs smaller automata and is able to scale better than `ltl2ldba` in these cases as well. The formulae are instantiations of: $h(m, n) = (\mathbf{X}^m p)\, \mathbf{U}\, (q \vee (\wedge_{i=1}^{n}(a_i\, \mathbf{U}\, \mathbf{X}^m b_i)))$.
5. The last few examples are from outside of LTL$\backslash GU$. The first three are in LTL$_\mathrm{D}$ while the rest are outside LTL$_\mathrm{D}$. We found that `Büchifier` did better only in a few cases (like ψ_3), this is due to the multiplicative effect that the internal untils have on the size of the automaton. So there is scope for improvement and we believe there are several optimizations that can be done to reduce the size in such cases and leave it for future work. $\psi_1 = \mathbf{FG}((a \wedge \mathbf{X}^2 b \wedge \mathbf{GF}b)\, \mathbf{U}\, (\mathbf{G}(\mathbf{X}^2 \neg c \vee \mathbf{X}^2(a \wedge b))))$, $\psi_2 = \mathbf{G}(\mathbf{F}\neg a \wedge \mathbf{F}(b \wedge \mathbf{X}c) \wedge \mathbf{GF}(a\, \mathbf{U}\, d))$, $\psi_3 = \mathbf{G}((\mathbf{X}^3 a)\, \mathbf{U}\, (b \vee \mathbf{G}c))$, $\psi_4 = \mathbf{G}((a\, \mathbf{U}\, b) \vee (c\, \mathbf{U}\, d))$, $\psi_5 = \mathbf{G}(a\, \mathbf{U}\, (b\, \mathbf{U}\, (c\, \mathbf{U}\, d)))$.

7 Conclusion

In this paper we presented a translation of formulas in LTL to limit deterministic automata, generalizing the construction from [8]. While the automata resulting from the translation can, in general, be doubly exponential in the size of the original formula, we observe that for formulas in the subfragment LTL$_\mathrm{D}$, the automaton is guaranteed to be only exponential in size. The logic LTL$_\mathrm{D}$ is a more expressive fragment than LTL$\backslash GU$, and thus our results enlarge the fragment of LTL for which small limit deterministic automata can be constructed. One consequence of our results here is a new EXPTIME algorithm for model checking MDPs against LTL$_\mathrm{D}$ formulas, improving the previously known upper bound of 2EXPTIME.

Our results in this paper, however, have not fully settled the question of when exponential sized limit deterministic automata can be constructed. We do not believe LTL$_\mathrm{D}$ to be the largest class. For example, our construction yields small automata for $\varphi = \mathbf{G}(\vee_i (p_i\, \mathbf{U}\, q_i))$, where p_i, q_i are propositions. φ is not expressible in LTL$_\mathrm{D}$. Of course we cannot have an exponential sized construction for full LTL as demonstrated by the double exponential lower bound in [20].

References

1. Büchifier. http://kini2.web.engr.illinois.edu/buchifier/
2. Alur, R., La Torre, S.: Deterministic generators and games for LTL fragments. ACM Trans. Comput. Logic **5**(1), 1–25 (2004)
3. Babiak, T., Blahoudek, F., Křetínský, M., Strejček, J.: Effective translation of LTL to deterministic Rabin automata: beyond the (F,G)-fragment. In: Hung, D., Ogawa, M. (eds.) ATVA 2013. LNCS, vol. 8172, pp. 24–39. Springer, Heidelberg (2013). doi:10.1007/978-3-319-02444-8_4
4. Courcoubetis, C., Yannakakis, M.: The complexity of probabilistic verification. J. ACM **42**(4), 857–907 (1995)
5. Esparza, J., Křetínský, J.: From LTL to deterministic automata: a safraless compositional approach. In: Biere, A., Bloem, R. (eds.) CAV 2014. LNCS, vol. 8559, pp. 192–208. Springer, Cham (2014). doi:10.1007/978-3-319-08867-9_13
6. Hahn, E.M., Li, G., Schewe, S., Turrini, A., Zhang, L.: Lazy probabilistic model checking without determinisation. In: CONCUR, pp. 354–367 (2015)
7. Henzinger, T.A., Piterman, N.: Solving games without determinization. In: Ésik, Z. (ed.) CSL 2006. LNCS, vol. 4207, pp. 395–410. Springer, Heidelberg (2006). doi:10.1007/11874683_26
8. Kini, D., Viswanathan, M.: Limit deterministic and probabilistic automata for LTL\GU. In: Baier, C., Tinelli, C. (eds.) TACAS 2015. LNCS, vol. 9035, pp. 628–642. Springer, Heidelberg (2015). doi:10.1007/978-3-662-46681-0_57
9. Kini, D., Viswanathan, M.: Optimal translation of LTL to limit deterministic automata. Technical report, University of Illinois at Urbana-Champaign (2017). http://hdl.handle.net/2142/95004
10. Klein, J., Baier, C.: Experiments with deterministic ω-automata for formulas of linear temporal logic. Theor. Comput. Sci. **363**(2), 182–195 (2006)
11. Klein, J., Müller, D., Baier, C., Klüppelholz, S.: Are good-for-games automata good for probabilistic model checking? In: Dediu, A.-H., Martín-Vide, C., Sierra-Rodríguez, J.-L., Truthe, B. (eds.) LATA 2014. LNCS, vol. 8370, pp. 453–465. Springer, Cham (2014). doi:10.1007/978-3-319-04921-2_37
12. Komárková, Z., Křetínský, J.: Rabinizer 3: safraless translation of LTL to small deterministic automata. In: Cassez, F., Raskin, J.-F. (eds.) ATVA 2014. LNCS, vol. 8837, pp. 235–241. Springer, Cham (2014). doi:10.1007/978-3-319-11936-6_17
13. Křetínský, J., Esparza, J.: Deterministic automata for the (F,G)-fragment of LTL. In: Madhusudan, P., Seshia, S.A. (eds.) CAV 2012. LNCS, vol. 7358, pp. 7–22. Springer, Heidelberg (2012). doi:10.1007/978-3-642-31424-7_7
14. Křetínský, J., Garza, R.L.: Rabinizer 2: small deterministic automata for LTL\GU. In: Hung, D., Ogawa, M. (eds.) ATVA 2013. LNCS, vol. 8172, pp. 446–450. Springer, Heidelberg (2013). doi:10.1007/978-3-319-02444-8_32
15. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011). doi:10.1007/978-3-642-22110-1_47
16. Morgenstern, A., Schneider, K.: From LTL to symbolically represented deterministic automata. In: Logozzo, F., Peled, D.A., Zuck, L.D. (eds.) VMCAI 2008. LNCS, vol. 4905, pp. 279–293. Springer, Heidelberg (2008). doi:10.1007/978-3-540-78163-9_24
17. Piterman, N., Pnueli, A., Sa'ar, Y.: Synthesis of reactive(1) designs. In: VMCAI, pp. 279–293 (2006)

18. Piterman, N., Pnueli, A., Sa'ar, Y.: Synthesis of reactive(1) designs. In: Emerson, E.A., Namjoshi, K.S. (eds.) VMCAI 2006. LNCS, vol. 3855, pp. 364–380. Springer, Heidelberg (2005). doi:10.1007/11609773_24
19. Puterman, M.L.: Markov Decision Processes. Wiley, Hoboken (1994)
20. Sickert, S., Esparza, J., Jaax, S., Křetínský, J.: Limit-deterministic Büchi automata for linear temporal logic. In: Chaudhuri, S., Farzan, A. (eds.) CAV 2016. LNCS, vol. 9780, pp. 312–332. Springer, Cham (2016). doi:10.1007/978-3-319-41540-6_17
21. Sickert, S., Křetínský, J.: MoChiBA: probabilistic LTL model checking using limit-deterministic Büchi automata. In: Artho, C., Legay, A., Peled, D. (eds.) ATVA 2016. LNCS, vol. 9938, pp. 130–137. Springer, Cham (2016). doi:10.1007/978-3-319-46520-3_9
22. Vardi, M., Wolper, P., Sistla, A.P.: Reasoning about infinite computation paths. In: FOCS (1983)
23. Vardi, M.Y.: Automatic verification of probabilistic concurrent finite-state programs. In: Proceedings of FOCS, pp. 327–338 (1985)