Model Checking Strategies from Synthesis Over Finite Traces

Suguman Bansal[1][0000−0002−0405−073X], Yong Li[2][0000−0002−7301−9234], Lucas M. Tabajara[3][0000−0001−9608−1404], Moshe Y. Vardi[4][0000−0002−0661−5773], and Andrew Wells[4][0000−0001−7780−2122]⋆

[1] Georgia Institute of Technology, GA, USA
[2] University of Liverpool, UK
[3] Runtime Verification, USA
[4] Rice University, TX, USA

Abstract. The innovations in reactive synthesis from *Linear Temporal Logics over finite traces* (LTLf) will be amplified by the ability to verify the correctness of the strategies generated by LTLf synthesis tools. This motivates our work on LTLf *model checking*. LTLf model checking, however, is not straightforward. The strategies generated by LTLf synthesis may be represented using *terminating* transducers or *non-terminating* transducers where executions are of finite-but-unbounded length or infinite length, respectively. For synthesis, there is no evidence that one type of transducer is better than the other since they both demonstrate the same complexity and similar algorithms.

In this work, we show that for model checking, the two types of transducers are fundamentally different. Our central result is that LTLf model checking of non-terminating transducers is *exponentially harder* than that of terminating transducers. We show that the problems are EXPSPACE-complete and PSPACE-complete, respectively. Hence, considering the feasibility of verification, LTLf synthesis tools should synthesize terminating transducers. This is, to the best of our knowledge, the *first* evidence to use one transducer over the other in LTLf synthesis.

1 Introduction

Linear Temporal Logic over finite traces [13] (LTLf) is the finite-horizon counterpart of the well-known Linear Temporal Logic (LTL) over infinite traces [23]. LTLf is rapidly gaining popularity among real-world applications where behaviors are better expressed over a finite but unbounded horizon [6, 10, 11, 18, 34].

Reactive synthesis from LTLf specifications, or LTLf synthesis [2, 7, 9, 12, 14, 17, 28, 36] has amassed so much interest that the 2023 Reactive Synthesis Competition (SYNTCOMP) will inaugrate an LTLf track[5]. Consequently, LTLf synthesis tools have been growing in complexity [2, 8, 17, 28, 36]. Their correctness, however, is rarely verified. To continue the innovations in synthesis and to

⋆ Work was performed while the author was at Rice University
[5] http://www.syntcomp.org/news/

successfully conduct large-scale competitions like SYNTCOMP there is, therefore, a need to verify the correctness of the synthesized strategies/transducers. Verifying the results as opposed to verifying the tools has been advocated in various contexts, including translation validation [26], program checking [5], and equivalence checking [21]. For LTL synthesis, result checking is simply LTL *model checking*. For LTLf synthesis, we need LTLf *model checking*. But this is a topic that has *not* been studied so far, hence this work.

We observe that LTLf model checking for LTLf synthesis tools is *not* as straightforward as one might have thought to be. The standard approach in the literature on LTLf synthesis generates *non-terminating transducers*. This includes the seminal work on synthesis [12] and the SYNTCOMP guidelines [19]. The executions of non-terminating transducers are of infinite length. Since LTLf formulas are defined on finite traces only, an execution of a non-terminating transducer is said to satisfy an LTLf formula if there *exists* a finite-length prefix that satisfies the formula [12]. Few works on synthesis do mention the possibility of *terminating transducers* as the output [2, 36]. Since their executions are of finite length, LTLf satisfaction is defined naturally on terminating transducers. When it comes to synthesis, there is no clear evidence that one type of transducer is better than the other, since the complexity and algorithms of synthesis are the same for both types. We believe this is why existing works on LTLf synthesis do *not* make a clear distinction between the two. For implementations, however, most works use non-terminating transducers as they directly correspond to standard Mealy/Moore machines (See state-of-the-art tools, e.g., Syft [36], Lisa [2], and Lydia [8]). This work shows, however, that from the *model-checking* perspective, the two types of transducers are *fundamentally different* and bear a significant impact on synthesis.

Our central result is that LTLf model checking of non-terminating transducers is *exponentially harder* than LTLf model checking of terminating transducers. We demonstrate that under LTLf specifications, model checking non-terminating transducers is EXPSPACE-complete, whereas model checking terminating transducers is PSPACE-complete. An immediate implication of this result is that for non-terminating transducers, LTLf model checking is exponentially harder than LTL model checking, which is known to be PSPACE-complete [32]. This result is unexpected because a factor behind the increasing popularity of LTLf is the perception that problems using LTLf are at most as hard as those using LTL, if not simpler (See Table 1). This is because LTLf formulas can be expressed by automata over finite words [13], which allow for practically scalable algorithms for automata constructions [29]. Conversely, LTL formulas require automata over infinite words [35], for which the automata manipulation is harder in theory [16, 25, 30, 31] and in practice [15, 20]. It is no wonder that an exponential increase in the model-checking complexity seems surprising at first.

The exponential blow-up in LTLf model-checking of non-terminating transducers arises from subtlety in the problem definition. A transducer satisfies a formula if there are no counterexamples. In non-terminating transducers, an infinite execution is a counterexample if *every* finite prefix does not satisfy the

Table 1: LTL vs. LTLf: Complexity w.r.t. specification. NT and T abbreviate non-terminating and terminating models, respectively.

	LTL	LTLf
Non-deterministic Automata	(NBA) Exponential	(NFA) Exponential
Satisfiability	PSPACE-complete [27]	PSPACE-complete [13]
Synthesis	2EXPTIME-complete [24]	2EXPTIME-complete [12]
Model Checking (NT)	PSPACE-complete [32]	EXPSPACE-complete (New!)
Model Checking (T)	Undefined	PSPACE-complete (New!)

LTLf formula. Formally, for an LTLf formula ϕ, let $\mathsf{pref}(\phi)$ represent the language consisting of all infinite executions for which every prefix satisfies ϕ. Then, a non-terminating transducer $\mathcal{M}$ satisfies an LTLf formula ϕ iff $\mathcal{L}(\mathcal{M}) \cap \mathsf{pref}(\neg\phi) = \emptyset$, where $\mathcal{L}(\mathcal{M})$ is the set of all executions of $\mathcal{M}$. This is where LTLf model checking fundamentally differs from LTL model checking, as counterexamples in LTL are obtained simply from an automaton for the negation of the formula [32]. W.l.o.g., we show that while $\mathsf{pref}(\phi)$ is ω-regular for all LTLf formulas ϕ, the size of their non-deterministic Büchi automata (NBA) is doubly exponential in the size of the formula, i.e., $2^{2^{\mathcal{O}(|\phi|)}}$ and $2^{2^{\Omega(\sqrt{|\phi|})}}$. Once again, this differs from LTL model checking, where the size of the NBAs for counterexamples is singly exponential in the size of the formula. As a result, we show LTLf model checking of non-terminating transducers is in EXPSPACE using on-the-fly emptiness checking of $\mathcal{L}(\mathcal{M}) \cap \mathsf{pref}(\neg\phi)$. We establish EXPSPACE-hardness from first principles.

In contrast, we show that LTLf model checking of terminating transducers is PSPACE-complete. Due to their finite-length executions, counterexamples in terminating transducers are completely characterized by the negation of the formula, lending the same complexity as LTL model checking.

Thus, our results offer a clear recommendation between the two types of transducers in LTLf synthesis. We argue that synthesis tools should account for the feasibility of the verification of the synthesized transducers. Consequently, we recommend that synthesis tools should generate terminating transducers rather than non-terminating transducers. We believe this is the *first* work to offer *theoretical* evidence to use one transducer over the other in synthesis. Furthermore, these results could be applied immediately to run the LTLf track in SYNTCOMP.

Outline. Section 2 outlines preliminaries on LTLf and LTLf synthesis. Section 3 motivates and defines LTLf model checking. Section 4 is dedicated to $\mathsf{pref}(\phi)$. Section 5 develops the complexity of model checking. Lastly, Section 6 concludes.

2 Preliminaries and Notations

We use the standard notions of deterministic and non-deterministic finite automata (DFAs and NFAs, respectively) as well as deterministic and non-deterministic Büchi automata (DBAs and NBAs, respectively). For an automaton, we use the notation $\mathcal{A} = (\Sigma, S, \iota, \delta, F)$ where Σ is a finite set of symbols

(called an alphabet), S is a finite set of states, $\iota \in S$ is the initial state, $F \subseteq S$ is the set of accepting states, and $\delta \subseteq S \times \Sigma \times S$ is the transition relation. We use standard semantics for all automata, hence defer details to the appendix.

2.1 Linear Temporal Logic over Finite Traces (LTLf)

LTLf [1, 13] extends propositional logic with finite-horizon temporal operators. In effect, LTLf is a variant of LTL [23] that is interpreted over finite rather than infinite traces. The syntax of an LTLf formula over a finite set of propositions Prop is identical to LTL, and defined as

$$\varphi := \mathsf{true} \mid \mathsf{false} \mid a \in \mathsf{Prop} \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \mathrm{X}\varphi \mid \varphi_1 \mathrm{U} \varphi_2$$

where X (Next) and U (Until), are temporal operators. We also include their dual operators, N (Weak Next) and R (Release), defined as $\mathrm{N}\varphi \equiv \neg\mathrm{X}\neg\varphi$ and $\varphi_1 \mathrm{R} \varphi_2 \equiv \neg(\neg\varphi_1 \mathrm{U} \neg\varphi_2)$. We also use typical abbreviations such as $\mathrm{F}\varphi \equiv \mathsf{true}\mathrm{U}\varphi$, $\mathrm{G}\varphi \equiv \mathsf{false}\mathrm{R}\varphi$, $\varphi_1 \vee \varphi_2 = \neg(\neg\varphi_1 \wedge \neg\varphi_2)$, $\varphi_1 \rightarrow \varphi_2 \equiv \neg\varphi_1 \vee \varphi_2$. We denote by $|\phi|$ the length/size of a formula ϕ, i.e., the number of operators in ϕ.

The semantics of LTLf is similar to LTL but is interpreted over finite traces. A finite sequence ρ over 2^{Prop} is said to satisfy an LTLf formula ϕ over Prop, denoted by $\rho \models \phi$, if $\rho, 0 \models \phi$ where for all positions $0 \leq i < |\rho|$, $\rho, i \models \phi$ is defined inductively on ϕ as follows:

- $\rho, i \models \mathsf{true}$; $\rho, i \not\models \mathsf{false}$; $\rho, i \models a$ iff $a \in \rho_i$
- $\rho, i \models \neg\varphi$ iff $\rho, i \not\models \varphi$
- $\rho, i \models \phi_1 \wedge \phi_2$ iff $\rho, i \models \phi_1$ and $\rho, i \models \phi_2$;
- $\rho, i \models \mathrm{X}\phi$ iff $i + 1 < |\rho|$ and $\rho, i + 1 \models \phi$
- $\rho, i \models \phi_1 \mathrm{U} \phi_2$ iff there exists j s.t. $i \leq j < |\rho|$ and $\rho, j \models \phi_2$, and for all k, $i \leq k < j$, we have $\rho, k \models \phi_1$

Observe that X requires that there *exists* a next position; In the context of *finite* traces, its negation also contains the situation that no next position exists, formulated as $\neg(\mathrm{X}\mathsf{true})$ or equivalently $\mathrm{N}\mathsf{false}$. This differs from LTL where the Next operator is applied to all positions. Also, note that LTLf formulas are evaluated on traces of non-zero length.

The language of an LTLf formula ϕ over Prop is the set of all finite sequences ρ over 2^{Prop} such that $\rho \models \phi$. The language of an LTLf formula is regular. The NFA and DFA representing LTLf are of size singly exponential and doubly exponential, respectively, in the size of the formula [13]. We note that a letter $\sigma \in \Sigma$ of the NFA/DFA corresponds to a valuation over the set Prop of propositions.

2.2 LTLf Synthesis and Transducers

Let LTLf formula ϕ be defined over propositional variables partitioned into $\mathcal{I}$ and $\mathcal{O}$ representing the input and output variables, respectively. Given such an LTLf formula ϕ, the problem of LTLf *realizability* is to determine whether there

exists a strategy $f : (2^{\mathcal{I}})^* \to 2^{\mathcal{O}}$ such that for all $\lambda_{\mathcal{I}} = I_0, I_1, \cdots \in (2^{\mathcal{I}})^\omega$, there is an integer $k \geq 0$ such that the finite trace $\rho = (I_0 \cup f(\varepsilon)), (I_1 \cup f(I_0)), \cdots, (I_k \cup f(I_0, I_1, \cdots, I_{k-1}))$ satisfies ϕ. The LTLf *synthesis problem* is to generate such a function, if the given formula is realizable [12]. Intuitively, LTLf synthesis can be viewed as a game between two agents, an environment and a system, who continually take turns to assign values to the input and output variables, respectively, to generate a sequence of input and output variables. W.l.o.g., we assume the system plays first, followed by the environment, and so on. The goal of synthesis is to generate a strategy for the system agent so that all resulting plays with the environment satisfy the given specification. We note that our model-checking results also hold when the environment plays first, as we will model strategies as transition systems in model checking for generality (cf. Section 3).

Non-terminating transducers. The standard in LTLf synthesis is to represent the strategy f using (non-terminating) transducers [12,19]. W.l.o.g., a transducer is a *Moore machine* $\mathcal{M} = (Q, q_0, \mathcal{I}, \mathcal{O}, \delta, G)$ where Q is a finite set of states, $q_0 \in Q$ is the initial state, and $\mathcal{I}$ and $\mathcal{O}$ are finite sets of input and output variables, respectively. Functions $\delta : Q \times 2^{\mathcal{I}} \to Q$ and $G : Q \to 2^{\mathcal{O}}$ are the *transition function* and the *output function*, respectively. Given an input sequence $\lambda_{\mathcal{I}} = I_0, I_1, \cdots \in (2^{\mathcal{I}})^\omega$, the output sequence is $\lambda_{\mathcal{O}} = G(q_0), G(q_1), \cdots \in (2^{\mathcal{O}})^\omega$ where q_0 is the initial state and $q_{i+1} = \delta(q_i, I_i)$ for all $i \geq 0$.

Then, given an LTLf formula with variables partitioned into $\mathcal{I}$ and $\mathcal{O}$ the realizability and synthesis problem is to generate a Moore machine $\mathcal{M}$ such that for all input sequences $\lambda = I_0, I_1, \cdots \in (2^{\mathcal{I}})^\omega$, there exists an integer $k \geq 0$ such that $\rho = (I_0, G(q_0)), (I_1, G(q_1)) \ldots (I_k, G(q_k))$ satisfies ϕ. Intuitively, the system and environment play indefinitely, where the system plays as per the transducer. The play (an execution in the transducer) satisfies an LTLf formula if there exists a finite-length prefix that satisfies the formula.

Terminating transducers. The strategy f can also be represented using terminating transducers [2, 36]. W.l.o.g., a terminating transducer is a *Terminating Moore machine* $\mathcal{M} = (Q, q_0, \mathcal{I}, \mathcal{O}, \delta, G, F)$ where Q, q_0, $\mathcal{I}$, $\mathcal{O}$, δ, and G are as defined for Moore machines and $\emptyset \neq F \subseteq Q$ are the *terminal states*. An input sequence $\lambda_{\mathcal{I}} = I_0, I_1, \cdots I_k \in (2^{\mathcal{I}})^*$ generates an output sequence $\lambda_{\mathcal{O}} = G(q_0), G(q_1), \ldots G(q_k) \in (2^{\mathcal{O}})^*$ where q_0 is the initial state and $q_{i+1} = \delta(q_i, I_i)$ for all $0 \leq i < k$.

Then, given an LTLf formula with variables partitioned into $\mathcal{I}$ and $\mathcal{O}$, the realizability and synthesis problem is to generate a terminating Moore machine $\mathcal{M}$ such that for all input sequence $\lambda = I_0, I_1, \cdots \in (2^{\mathcal{I}})^\omega$, there exists an integer $k \geq 0$ such that $\rho = (I_0, G(q_0)), (I_1, G(q_1)) \ldots (I_k, G(q_k))$ with $q_{k+1} = \delta(q_k, I_k) \in F$ and ρ satisfies ϕ. Intuitively, the synthesized terminating transducer is such that as soon as a play lands in a terminal state of the transducer, the system agent controlling the output variables wins the game and this play is over as it is guaranteed that the play seen so far satisfies the given formula. On the contrary, in non-terminating transducers, the system agent does not have the ability to

terminate a game as it is never informed of whether it has seen a satisfying prefix.

3 LTLf Model Checking

In addition to being of independent interest, our motivation behind LTLf model checking is to support the ongoing development of LTLf synthesis tools. As synthesis tools continue to become more complex, it is imperative that we design automatic approaches to check their correctness. One way is to evaluate whether the result generated from these tools is correct. In the case of LTLf synthesis, result checking corresponds to LTLf model checking. Finally, an immediate application of LTLf model checking could be in running the inaugural LTLf track in the Reactive Synthesis Competition (SYNTCOMP) [19].

We begin by defining the model-checking problem. As described in Section 2.2, the result of LTLf synthesis could be a terminating or a non-terminating transducer. Since LTLf satisfaction on executions in the two types of transducers differ, we define model-checking on them separately. For the sake of generality, we define model-checking with respect to *transition systems* (TS) as opposed to transducers. Translations from transducers to transition systems are standard and polynomial [22]. Hence, the translation details have been omitted.

Non-Terminating Transition Systems are those that run indefinitely, i.e., their executions are of infinite length (e.g. network servers). Formally, a non-terminating TS is a structure $\mathcal{M} = (\Sigma, S, T, \iota, L)$, where Σ is a finite propositional alphabet, S is a finite set of states, relation $T \subseteq S \times S$ is the transition relation with no sink states, ι is the initial state, and $L : S \to 2^{\Sigma}$ is the *labeling function*. An *execution* $\rho = s_0 s_1 \cdots$ in $\mathcal{M}$ is an infinite sequence of consecutive states beginning with the initial state, i.e., $s_0 = \iota$ and $(s_i, s_{i+1}) \in T$ for all $i \geq 0$. The *label sequence* of ρ is the sequence $L(\rho) = L(s_0)L(s_1)\cdots$. The n-length finite prefix of ρ and its label sequence are given by $\rho[0, n] = s_0 \cdots s_{n-1}$ and $L(\rho[0, n]) = L(s_0) \cdots L(s_{n-1})$, respectively, for $n > 0$.

Since executions are of infinite-length and LTLf formulas are interpreted over finite-length sequences only, we say an *execution ρ in $\mathcal{M}$ satisfies an* LTLf *formula ϕ*, denoted by $\rho \models \mathcal{M}$, as follows

$$\rho \models \phi \text{ iff } \exists n > 0 \text{ s.t. } L(\rho[0, n]) \models \phi,$$

i.e., there exists a finite-length prefix of the execution that satisfies the formula.

Terminating Transition Systems are those that terminate after a finite but unbounded amount of steps (e.g. a terminating program). Formally, a terminating TS is given by a structure $\mathcal{M} = (\Sigma, S, T, \iota, L, F)$, where Σ, S, $T \subseteq S \times S$, ι, and $L : S \to 2^{\Sigma}$ are defined as for nonterminating transition systems and $\emptyset \neq F \subseteq S$ are the *terminal states*, which are the only states that are allowed to be sink states. An *execution* $\rho = s_0 \cdots s_n$ in $\mathcal{M}$ is a finite sequence of consecutive states beginning with the initial state and ending in a terminal state, i.e., $s_0 = \iota$ and

$(s_i, s_{i+1}) \in T$ for all $0 \leq i < n$, and $s_n \in F$. Its *label sequence* is the sequence $L(\rho) = L(s_0) \cdots L(s_n)$.

An *execution ρ in $\mathcal{M}$ satisfies an* LTLf *formula ϕ*, denoted by $\rho \models \phi$,

$$\rho \models \phi \text{ iff } L(\rho) \models \phi.$$

Model Checking. We first define *satisfaction* and then *model checking.*

Definition 1 ($\mathcal{M} \models \phi$). *Given a non-terminating (resp., terminating) transition system $\mathcal{M}$ and an* LTLf *formula ϕ, we say TS $\mathcal{M}$* satisfies *ϕ, denoted by $\mathcal{M} \models \phi$, if for all (resp., finite) executions ρ of $\mathcal{M}$, we have that $\rho \models \phi$.*

Definition 2 (Model Checking). *Given a non-terminating (resp. terminating) transition system $\mathcal{M}$ and an* LTLf *formula φ, the problem of* LTLf *model checking of non-terminating (resp. terminating) models is to determine whether $\mathcal{M}$ satisfies φ.*

Note on abuse of notation. The notation $\models$ has been overloaded to express satisfaction at several occasions, namely, in LTLf semantics, in defining when executions of non-terminating and terminating systems satisfy a formula, and when a system satisfies a formula. We overload notation to avoid new symbols for each case, as the context is clear from the L.H.S.

4 Prefix Language of LTLf Formulas

This section builds the basic blocks for LTLf model checking of non-terminating systems. Recall from Section 3, an (infinite-length) execution in a non-terminating system $\mathcal{M}$ violates an LTLf formula ϕ if *all* of its finite prefixes violate ϕ. So, the counterexamples are captured by the language that accepts an infinite word iff all of its finite prefixes violate ϕ (or satisfy $\neg\phi$). We call this the *prefix language* of an LTLf formula $\neg\phi$. Then, clearly, $\mathcal{M} \models \phi$ iff the intersection of $\mathcal{M}$ with the prefix language of $\neg\phi$ is empty, making the prefix language a basic block to model-check non-terminating systems.

We first observe that the prefix languages for LTLf formulas are ω-regular. We then show that one can construct a DBA accepting the prefix language of an LTLf formula, which incurs a doubly exponential blow-up (Section 4.1). One may expect that the complexity of the construction can be improved if we target at NBAs. We show, however, that the doubly exponential blow-up is *not* due to a lack of better construction, but a fundamental trait of the problem itself (Theorem 2). This is in contrast to the construction of NBA/NFA for LTL/ LTLf, where only deterministic automata constructions incur doubly exponential blow-ups and nondeterministic automata constructions incur singly exponential blow-ups, hinting at the hardness of model checking. Finally, we identify a fragment of LTLf formulas for which a singly exponential construction of NBAs for their prefix languages can be obtained via a translation from LTLf to LTL (Section 4.2).

4.1 Prefix Automata for LTLf

This section formally defines the prefix language/automata for LTLf formulas and proves that their automata constructions involve an unavoidable double-exponential blow-up. The upper and lower bounds are shown in Theorem 1 and Theorem 2, respectively.

Definition 3 (Prefix Language). *Given an* LTLf *formula* ϕ*, the* prefix language of ϕ*, denoted by* $\mathsf{pref}(\phi)$*, is such that an (infinite-length) word* $w \in \mathsf{pref}(\phi)$ *iff every finite prefix of* w *satisfies* ϕ*, i.e.,* $\forall n > 0.w[0,n] \models \phi$*.*

Recall that the semantics of LTLf requires traces of non-zero length only (see Section 2). So we only need $n > 0$, instead of $n \geq 0$, ignoring the empty word. By abuse of notation, we let $\mathsf{pref}(\phi)$ denote both the prefix language and its corresponding automaton, called the *prefix automaton*.

We start by showing $\mathsf{pref}(\phi)$ is ω-regular for LTLf formula ϕ:

Theorem 1 (Prefix automata: Upper bound). *For an* LTLf *formula* ϕ*, the language* $\mathsf{pref}(\phi)$ *is* ω*-regular. The Büchi automaton recognizing* $\mathsf{pref}(\phi)$ *has* $2^{2^{\mathcal{O}(|\phi|)}}$ *states.*

Proof. Given LTLf formula ϕ, we construct a DBA for $\mathsf{pref}(\phi)$ as follows:

1. Construct a DFA $D = (\Sigma, Q, \iota, \delta, F)$ for $\neg\phi$, i.e., $\mathcal{L}(D) = \mathcal{L}(\neg\phi)$.
   We require D to be *complete* in the sense that for every state s and every alphabet $a \in \Sigma$, there exists a successor $t = \delta(s, a)$.
2. Obtain a DBA $C = (\Sigma, Q, \iota, \delta', F)$ by converting all accepting states F of D to accepting sink states in C. For this, replace all outgoing transitions from all accepting states in D with self loops on all letters.
   Formally, replace every $\delta(f, a) = t$ in DFA D with $f = \delta'(f, a)$ in DBA C, for all $f \in F$ and $a \in \Sigma$. For all other states, let δ' behaves identically to δ.
3. Obtain the desired Büchi automaton $B = (\Sigma, Q, \iota, \delta', \mathcal{F} = Q \setminus F)$ by swapping accepting and non-accepting states of C.

Since C is a DBA with accepting sink states, C is the complement of B. Hence, it suffices to show that C accepts $w \in \Sigma^\omega$ iff there exists a finite prefix of w that satisfies $\neg\phi$. Clearly, $w \in \mathcal{L}(C)$ then w must have a finite-prefix satisfying $\neg\phi$ since the accepting states of C and D are identical. Conversely, we need to show that despite δ and δ' being different, C will accept all words that contain a finite prefix satisfying $\neg\phi$. For this, we show that for every such word, C retains the transitions to accept the shortest prefix satisfying $\neg\phi$. Details can be found in the appendix. Finally, the number of states of C are bounded by those of D which is doubly exponential in $|\phi|$ [13]. □

Observe that the Büchi automaton B constructed above is *deterministic*. One of our key discoveries is that the doubly exponential blow-up appears even in the construction of NBAs for $\mathsf{pref}(\phi)$, demonstrating that the blow-up is fundamentally unavoidable. Theorem 2 presents such an LTLf formula to demonstrate the blow-up. The rest of the section builds up to that construction.

We observe that the blow-up is caused by the combination of two aspects: First is the universal quantification on prefixes of words in $\mathsf{pref}(\phi)$; Second is the ability of an LTLf formula to identify the k-th last positions of finite words using the X (Next) modality. At first, we identify an ω-regular language, parameterized with $n \geq 1$, such that all NBAs accepting the language have at least 2^{2^n} states. Let $n \in \mathbb{N}$ and $\Sigma = \{0, 1, \#, \&\}$. Consider the language $L_n \subseteq \Sigma^\omega$ where

$$u \cdot \& \cdot v \in L_n \text{ s.t. if } \#w\# \text{ appears in } v \text{ then } \#w\# \text{ also appears in } u,$$

where $w \in \{0,1\}^n$, $u \in \{0,1,\#\}^*$ and $v \in \{0,1,\#\}^\omega$. Intuitively, L_n consists of infinite words that are (a) split into two parts by a special character "&" and (b) all words of the form $\#w\#$ appearing after "&" must have appeared before "&", for all n-length words $w \in \{0,1\}^n$. Essentially, L_n is a bit-level adaption of the language K_d where $x \cdot \& \cdot y \in K_d$ if digits appearing in y are a subset of digits appearing in x, where $x \in D^*$ and $y \in D^\omega$ for $D = \{0, 1, \cdots, d-1\}$. Obviously, the words 14&1 and 134&4 are good prefixes of a word $x \cdot \& \cdot y \in K_d$ when $d > 5$. There are also less obvious good prefixes, such as a permutation of D followed by the letter &. We need to recognize all good prefixes in order to accept the language K_d. So, it is necessary to keep track of the digits (i.e., subsets of D) that the automaton has seen so far in an input word. Hence, the NBA of K_d needs $2^{\Omega(d)}$ states. The same proof can be adapted to show that the NBA of L_n consists of $2^{2^{\Omega(n)}}$ states. We defer a full proof to the supplemental material.

Next, we need to identify a regular language F_n such that, by abuse of notation, $\mathsf{pref}(F_n)$ corresponds to L_n and F_n can be represented by an LTLf formula of polynomial length in the parameter $n > 0$. A natural choice would be to let F_n to be the finite-word version of L_n. In other words, $u \cdot \& \cdot v \in F_n$ s.t. if $\#w\#$ appears in v then $\#w\#$ must have appeared in u for all $w \in \{0,1\}^n$ and $u, v \in \{0,1,\#\}^*$. The issue is that F_n cannot be represented by a short LTLf formula for the same reason why L_n cannot be expressed by a short LTL formula.

We need F_n to be a *simpler* language. The roadmap would be to leverage the universal quantification over all prefixes to generate L_n. This is also where we leverage the ability of LTLf to refer to the last k-th positions of a finite trace. Keeping these goalposts, we define regular language $F_n \subseteq \Sigma^*$ as

$$u \cdot \& \cdot v \in F_n \text{ s.t. if the last } n+2 \text{ characters of } v \text{ are of the form } \#w\# \\ \text{then } \#w\# \text{ also appears in } u,$$

where $w \in \{0,1\}^n$ and $u, v \in \{0,1,\#\}^*$. Intuitively, by applying universal quantification on all finite-length prefixes, focusing on the last $n+2$ characters of words in F_n is sufficient to ensure that every occurrence of the form $\#w\#$ after the symbol "&" appears in the portion before the "&".

There is one last caveat. There are infinitely many prefixes of words in L_n that may not contain the symbol &. This issue can be easily remedied by including words without symbol & to both languages. We overload the notation of $\mathsf{pref}(L)$ to refer to the prefix language of a language over finite words L. Then,

Lemma 1. *Let L_n and F_n be as defined above. Then*

$$L_n \uplus \{0,1,\#\}^\omega = \mathsf{pref}(F_n \uplus \{0,1,\#\}^*).$$

Proof (Proof Sketch). To see why $L_n \uplus \{0,1,\#\}^\omega \subseteq \mathsf{pref}(F_n \uplus \{0,1,\#\}^*)$, observe that the prefixes of a word $w \in L_n \uplus \{0,1,\#\}^\omega$ either contain the symbol & or they don't. If the prefix falls under the latter, then the prefix is contained in $\{0,1,\#\}^*$. Otherwise, if the last $n+2$ characters are not in the form $\#w\#$ for $w \in \{0,1\}^n$ then the prefix is contained in F_n by definition of F_n. If the last $n+2$ characters are in form $\#w\#$ for $w \in \{0,1\}^n$, then, by properties of words in L_n, $\#w\#$ must have appeared before &. Once again, the prefix is contained in F_n. Thus, all prefixes of w are contained in $F_n \uplus \{0,1,\#\}^*$.

The converse, i.e., $\mathsf{pref}(F_n \uplus \{0,1,\#\}^*) \subseteq L_n \uplus \{0,1,\#\}^\omega$, can be proven by a similar case-by-case analysis. Details can be found in the appendix. □

The last piece is to show that the language $F_n \uplus \{0,1,\#\}^*$ can be expressed using an LTLf formula ϕ_n of length polynomial in n, as shown below:

Theorem 2 (Prefix automata: Lower bound). *There exists an* LTLf *formula ψ such that the number of states in all NBAs for* $\mathsf{pref}(\psi)$ *is* $2^{2^{\Omega(\sqrt{|\psi|})}}$.

Proof. Let $n \in \mathbb{N} \setminus \{0\}$ and $\Sigma = \{0,1,\#,\&\}$. Let L_n and F_n be as defined above.

Since all NBAs of L_n are of size $2^{2^{\Omega(n)}}$ and L_n is disjoint from $\{0,1,\#\}^\omega$ by containing the "&" symbol, it is easy to show that all NBAs of $L_n \uplus \{0,1,\#\}^\omega$ require $2^{2^{\Omega(n)}}$ states as well.

From Lemma 1, it is sufficient to show that $F_n \uplus \{0,1,\#\}^*$ can be represented by an LTLf formula of length $\mathcal{O}(n^2)$. So, let us construct the desired LTLf formula ϕ_n. By abuse of notation, let the propositions be given by $\mathsf{Prop} = \{0,1,\#,\&\}$ with the interpretation that the symbol holds when its proposition is true. Recall that a letter σ in the finite alphabet Σ corresponds to a valuation over the atomic propositions Prop. For instance, $\& \in \Sigma$ is interpreted as the valuation $\neg 0 \wedge \neg 1 \wedge \neg \# \wedge \&$ over Prop. Then, the LTLf formula ϕ_n is a conjunction of the following three:

(R1). At all times, only one proposition can be true.
(R2). If "&" holds at some place, it occurs exactly once.
(R3). If "&" holds at some place, then if the end of the word has the form $\#w\#$, for $w \in \{0,1\}^n$, $\#w\#$ must have appeared before "&".

The LTLf formulation of (R1), denoted by OnlyOneProp, is quite straightforward and has been deferred to the supplementary material. The formulation of (R2) is F& → ExactOne&, where ExactOne& expresses that "&" occurs exactly once:

$$\mathsf{ExactOne\&} := (\neg\&\mathrm{U}(\& \wedge (\neg(\mathrm{X}\mathsf{true}) \vee \mathrm{X}(\mathrm{G}\neg\&)))).$$

Intuitively, the "&" symbol is not seen *until* it is seen somewhere, after which either the trace *terminates* (i.e., ¬(Xtrue) holds) or the trace does not see "&" *globally* (i.e., X(G¬&) holds). In fact, we also have $\neg(\mathrm{X}\mathsf{true}) \vee \mathrm{X}(\mathrm{G}\neg\&) \equiv \mathrm{N}(\mathrm{G}\neg\&)$.

To express (R3), we first introduce two formulas. The first is $\mathsf{EndWith\#w\#}$ to express that the end of the word has the form $\#w\#$. The second is $\mathsf{End\#w\#AppearsBefore\&}$ to express that the word $\#w\#$ must appear before "&". So, (R3) is expressed by

$$\mathrm{F}\& \rightarrow (\mathsf{EndWith\#w\#} \rightarrow \mathsf{End\#w\#AppearsBefore\&})$$

For $\mathsf{EndWith\#w\#}$, we introduce shorthands, namely $\mathsf{Ends} := \mathrm{X}^{n+1}(\neg(\mathrm{X}\mathsf{true}))$, and $\mathsf{Appear\#w\#} := \# \wedge \mathrm{X}^{n+1}\# \wedge \bigwedge_{i=1}^{n} \mathrm{X}^i(0 \vee 1)$. Note that Ends is true only at the $(n+2)$-th last position of a trace and $\mathsf{Appear\#w\#}$ enforces that the current and next $n+1$ positions have the form $\#w\#$ for $w \in \{0,1\}^n$. Then,

$$\mathsf{EndWith\#w\#} := \mathrm{G}(\mathsf{Ends} \rightarrow \mathsf{Appear\#w\#})$$

Also, $\mathsf{End\#w\#AppearsBefore\&} :=$

$$\mathrm{F}\Big(\mathsf{Appear\#w\#} \wedge \mathrm{F}\& \wedge \bigwedge_{i=1}^{n} [(\mathrm{X}^i 0 \wedge \mathrm{G}(\mathsf{Ends} \rightarrow \mathrm{X}^i 0)) \vee (\mathrm{X}^i 1 \wedge \mathrm{G}(\mathsf{Ends} \rightarrow \mathrm{X}^i 1))]\Big)$$

Intuitively, when defining $\mathsf{End\#w\#AppearsBefore\&}$, we assume that we are standing at the first position of a word of the form $\#w\#$ that appears before "&". So, we require that $\mathsf{Appear\#w\#}$ holds and later F& holds. Next, we require the same word w to appear at the end. So we require that if in the i-th position, 0 (resp. 1) holds, at the i-th position from where Ends holds, 0 (resp. 1) must also hold. This is formulated as $(\mathrm{X}^i 0 \wedge \mathrm{G}(\mathsf{Ends} \rightarrow \mathrm{X}^i 0)) \vee (\mathrm{X}^i 1 \wedge \mathrm{G}(\mathsf{Ends} \rightarrow \mathrm{X}^i 1))$.

Finally, the whole formula ϕ_n is given as follows:

$$\begin{aligned}\phi_n = {} & \mathsf{OnlyOneProp} \\ & \wedge (\mathrm{F}\& \rightarrow (\mathsf{ExactOne\&} \wedge ((\mathsf{EndWith\#w\#} \rightarrow \mathsf{End\#w\#AppearsBefore\&}))))\end{aligned}$$

Clearly, when F& does not hold, all words satisfying ϕ_n would be in $\{0,1,\#\}^\omega$. If F& holds, then all words should meet (R2) and (R3). One can easily verify that ϕ_n specifies the language $F_n \uplus \{0,1,\#\}^*$. Thus, $\mathsf{pref}(\phi_n) = L_n \uplus \{0,1,\#\}^\omega$.

Last but not the least, the length of ϕ_n is in $\mathcal{O}(n^2)$ since $\mathsf{End\#w\#AppearsBefore\&}$ has length of $\mathcal{O}(n^2)$. □

Note that the LTLf formulation makes heavy use of Ends, which in turn uses the X modality. Essentially, Ends serves as a unique identifier of a specific position at the end of all traces. This enables us to anchor at that location without any artificial constructs and to express the desiderata accordingly. This is a crucial difference between LTLf and LTL.

4.2 Prefix automata for LTLf Fragment

In this section, we show that a singly exponential construction of NBAs is possible for a fragment of LTLf formulas. Through an exposition of the prefix language

for fragments of LTLf, we highlight some of the peculiarities of the prefix language. Consider the fragment of LTLf, denoted as $\mathsf{LTLf}_{\setminus\{\mathrm{R},\vee\}}$, which permits all but the R (Release) modality and allows $\neg$ and $\vee$ on literals only, as defined below:

$$\psi := \ell \mid \neg\ell \mid \psi \wedge \psi \mid \mathrm{X}\psi \mid \mathrm{N}\psi \mid \mathrm{F}\psi \mid \mathrm{G}\psi \mid \psi\mathrm{U}\psi$$

where $\ell := a \in \mathsf{Prop} \mid \neg a \mid \ell \wedge \ell \mid \ell \vee \ell$. We show that the prefix language of this fragment is equivalently represented by an LTL formula of the same size, hence its NBA is singly exponential in the size of the formula. The said LTL formula can be obtained using the translation $t : \mathsf{LTLf}_{\setminus\{\mathrm{R},\vee\}} \to \mathsf{LTL}$ described below (Since LTL and LTLf share the same syntax, to avoid confusion, we add the subscript ∞ to temporal operators in LTL, indicating that we have $|\rho| = \infty$. For instance, Globally in LTL becomes G_∞):

- $t(\ell) = \ell$, $t(\neg\ell) = \neg\ell$
- $t(\mathrm{X}\psi) = \mathsf{false}$, $t(\mathrm{N}\psi) = \mathrm{X}_\infty t(\psi)$
- $t(\psi_1 \wedge \psi_2) = t(\psi_1) \wedge t(\psi_2)$
- $t(\mathrm{F}\psi) = t(\psi)$
- $t(\psi_1 \mathrm{U} \psi_2) = t(\psi_2)$
- $t(\mathrm{G}\psi) = \mathrm{G}_\infty(t(\psi))$

The insight behind this translation is to identify that the criteria for a formula to hold on all finite-length prefixes simplifies to the formula holding on a prefix of length one. The proof is presented below:

Lemma 2. *Let $\phi \in \mathsf{LTLf}_{\setminus\{R,\vee\}}$ and let LTL $t(\phi)$ be as defined above. Then, $\mathcal{L}(t(\phi)) = \mathsf{pref}(\phi)$ and $\mathcal{O}(|\phi|) = \mathcal{O}(|t(\phi)|)$.*

Proof. Trivially, $\mathcal{O}(|\phi|) = \mathcal{O}(|t(\phi|)$ holds. We prove that $\mathcal{L}(t(\phi)) = \mathsf{pref}(\phi)$ by structural induction on ϕ. In the interest of space, we skip the base cases (ℓ and $\neg\ell$). We also skip the $\wedge$ and G modalities, as they are intuitive. We present the argument for X, N, F, and U. The full proof has been deferred to the appendix.

We set up notations: for $w = w_0 w_1 \cdots \in \Sigma^\omega$, let $w[i,j] = w_i \cdots w_{j-1}$ denote subsequences of w for $0 \leq i < j$. So, $w[0,n]$ is the n-length prefix of w for $n > 0$. By inductive hypothesis (I.H.), we assume $\mathcal{L}(t(\gamma)) = \mathsf{pref}(\gamma)$ for $\gamma \in \{\psi, \psi_1, \psi_2\}$.

Case $\mathbf{F}\psi$: The critical observation is that for $\mathrm{F}\psi$ to hold on all finite prefixes, $\mathrm{F}\psi$ must hold on the prefix of length 1, which in turn is possible only if the first position of the word satisfies ψ. Formally, first we show that $\mathsf{pref}(\mathrm{F}\psi) \subseteq \mathcal{L}(t(\mathrm{F}\psi))$. Let $w \in \mathsf{pref}(\mathrm{F}\psi)$. Then, in particular $w[0,1] \models \mathrm{F}\psi$. This is possible only if $w[0,1] \models \psi$. Thus, for all $n > 0$, we get $w[0,n] \models \psi$. So, $w \in \mathsf{pref}(\psi)$. By I.H., $w \in \mathcal{L}(t(\psi))$. By translation, this means $w \in \mathcal{L}(t(\mathrm{F}\psi))$. Next, we show $\mathcal{L}(t(\mathrm{F}\psi)) \subseteq \mathsf{pref}(\mathrm{F}\psi)$. Let $w \in \mathcal{L}(t(\mathrm{F}\psi))$. By translation, $w \in \mathcal{L}(t(\psi))$. By I.H., $w \in \mathsf{pref}(\psi)$. Now, if ψ holds, then $\mathrm{F}\psi$ also holds for all non-zero lengths. Hence, $w \in \mathsf{pref}(\mathrm{F}\psi)$.

Case $\psi_1\mathbf{U}\psi_2$: As earlier, the critical observation is for $\psi_1\mathrm{U}\psi_2$ to hold on a prefix of length one. For this, ψ_2 must hold. The proof is similar to the earlier case.

Case $\mathbf{X}\psi$: The issue is that $\mathrm{X}\psi$ can never be true on a word of length one, since there does not exist a next position on length one words. Hence, $\mathsf{pref}(\mathrm{X}\psi) = \emptyset = \mathcal{L}(\mathsf{False}) = \mathcal{L}(t(\mathrm{X}\psi))$.

Case Nψ: N (Weak Next) doesn't have the issue faced by X. If a word is of length one, Nψ trivially holds. For words of all other lengths, it requires Xψ to hold. Formally, first we show that $\mathsf{pref}(\mathrm{N}\psi) \subseteq \mathcal{L}(t(\mathrm{N}\psi))$. Let $w \in \mathsf{pref}(\mathrm{N}\psi)$. Then, by semantics of LTLf, it follows that the second position on w must satisfy ψ, i.e., $w[1,2] \models \psi$. In particular, for all $i > 1$, $w[1,i] \models \psi$. So, $w[1,\infty] \in \mathsf{pref}(\psi)$. By I.H., $w[1,\infty] \in \mathcal{L}(t(\psi))$. Hence, $w \in \mathcal{L}(\mathrm{X}_\infty t(\psi)) = \mathcal{L}(t(\mathrm{N}\psi))$. Conversely, let $w \in \mathcal{L}(t(\mathrm{N}\psi))$. By translation, $w \in \mathcal{L}(\mathrm{X}_\infty t(\psi))$. Hence, by I.H., we get for all $i > 1$, $w[0,i] \models \mathrm{X}\psi$ and $w[0,1] \models \mathrm{N}\psi$ since $w[1,\infty] \in \mathcal{L}(t(\psi)) = \mathsf{pref}(\psi)$. In other words, $w \in \mathsf{pref}(\mathrm{N}\psi)$.

□

An immediate consequence of Lemma 2 is that the prefix automata for $\mathsf{LTLf}_{\setminus\{\mathrm{R},\vee\}}$ are singly exponential in the size of the formula [33]:

Corollary 1. *Let $\phi \in \mathsf{LTLf}_{\setminus\{R,\vee\}}$. The NBA for $\mathsf{pref}(\phi)$ contains $2^{\mathcal{O}(|\phi|)}$ states.*

Note that, in all the cases above, every conjunct holds on *all* finite prefixes. This may not be true if $\vee$ (or) is permitted in the formula. For example, consider $\phi = \mathrm{G}a \vee \mathrm{F}b$. Now, the word $w = \{a\}\{b\}\{\}^\omega \in \mathsf{pref}(\phi)$ since the prefix of length one satisfies Ga and all other prefixes satisfy Fb. Hence, with disjunction, different prefixes can satisfy *different* disjuncts. In fact, the LTL formula for $\mathsf{pref}(\phi)$ is $a\mathrm{U}_\infty b \vee \mathrm{G}_\infty a$. However, such translations may increase the formula length because of duplicating the formula under G_∞ modality. An open problem here is to identify the largest fragment for which the prefix automata have only singly exponential blow-up. This goes hand-in-hand with uncovering the core behind the doubly exponential blow-up for prefix automata.

5 Complexity of LTLf Model Checking

We present the complexity of LTLf model checking. Section 5.1 develops the lower bound for model checking non-terminating systems and Section 5.2 presents the completeness argument for both terminating and non-terminating systems.

5.1 EXPSPACE Lower Bound for Non-terminating Systems

We prove EXPSPACE-hardness of LTLf model checking of non-terminating systems by a polynomial-time reduction from the problem of whether an exponential-space Turing machine $T = (Q, \Gamma, \delta, q_0, F)$ accepts an input word $x = x_1 \dots x_n$. The components of the Turing machine are defined as follows:

- Q is the set of states and $q_0 \in Q$ is the initial state.
- Γ is the tape alphabet, which is assumed to include the blank symbol $\emptyset$.
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{\leftarrow, \rightarrow\}$ is the transition function. $\delta(q,\gamma) = (q', \gamma', d)$ means that if the machine is in state q and the head reads symbol γ, it moves to state q', writes symbol γ', and moves the head in direction d.
- $F \subseteq Q$ is the set of accepting states. The machine accepts if it reaches a state in F.

Since T is an exponential-space Turing machine, we can assume that its tape has 2^{cn} cells, where n is the size of the input and c is a constant.

High-Level Idea Given a Turing machine T and an input x, our reduction will construct a non-terminating system M and an LTLf formula φ s.t. T accepts x iff every execution of M has a finite prefix that satisfies φ, i.e., $M \models \varphi$.

In this reduction, we will encode runs of the Turing machine as label sequences of the system. A *cell* in the tape is encoded as a sequence of $cn + 1$ propositional assignments. The first assignment encodes the content of the cell, which can be either a symbol $\gamma \in \Gamma$ or a symbol γ along with a state $q \in Q$, the latter indicating that the head is on that cell and is in state q. The remaining cn assignments encode the position of the cell in the tape as a cn-bit number (since the tape has 2^{cn} cells). The concatenation of 2^{cn} cells encodes a *configuration* of the Turing machine. Therefore, each configuration is encoded by $2^{cn}(cn + 1)$ assignments in total. The concatenation of configurations encodes a *run* of the Turing machine. Note, however, that for such a run to be consistent with the run of T on x, certain consistency conditions must hold:

1. For every configuration, the encoding of the position of the first cell must be 0, and the encoding must increase by 1 for each successive cell.
2. The first configuration must start with x on the tape and the head on the first cell and in the initial state q_0.
3. Successive configurations must be consistent with the transition function δ.

One way is to enforce all consistency conditions through the system M. However, since each configuration consists of 2^{cn} cells, this would require the system to have an exponential number of states. Therefore, to allow for a polynomial reduction, we enforce the consistency conditions through the formula φ.

For this, we construct an LTLf formula $\varphi := \varphi_{cons} \rightarrow \varphi_{acc}$. where φ_{cons} expresses the the consistency conditions and φ_{acc} expresses the property of reaching an accepting configuration. Therefore, every execution with a finite prefix that satisfies φ is either inconsistent or an accepting run of T on x. Since T is deterministic, there is exactly one execution of M that is consistent with T. Every other execution will necessarily satisfy $\neg\varphi_{cons}$, and this execution will satisfy φ_{acc} if and only if T accepts x. Therefore, if every execution of M has a finite prefix that satisfies φ, then the run of T on input x is accepting, and vice-versa.

We now provide the details of the system M and the formula φ.

Atomic Propositions The propositions used by system M are the following:

- $part_0$ indicates that the current assignment represents the first part of the cell encoding, encoding the cell's content.
- $part_i$, for $1 \leq i \leq cn$, indicates that the current assignment represents the i-th bit of the encoding of the cell's position. Only one of $part_0, \ldots, part_{cn}$ is true at any given time.

- $cell_\lambda$, for $\lambda \in \Gamma \cup (Q \times \Gamma)$, indicates that the content of the cell is λ (a tape symbol with or without the head). This proposition can only be true if $part_0$ is true.
- *bit* gives the current bit of the cell's position. This proposition can only be true if $part_0$ is false.

The Model We define the transition system $M = (\Sigma, S, T, \iota, L)$ as follows:

- $\Sigma = \{part_0, \ldots, part_{cn}\} \cup \{cell_\lambda \mid \lambda \in \Gamma \cup (Q \times \Gamma)\} \cup \{bit\}$
- $S = \{(0, \lambda) \mid \lambda \in \Gamma \cup (Q \times \Gamma)\} \cup \{(i, b) \mid 1 \leq i \leq cn, b \in \{0, 1\}\}$
- $\iota = (0, (q_0, \emptyset))$
- $(s, s') \in T$ if and only if one of the following is true (for some λ, b, b'):
  - $s = (0, \lambda)$ and $s' = (1, b)$.
  - $s = (i, b)$ for $1 \leq i < cn$, and $s' = (i + 1, b')$.
  - $s = (cn, b)$ and $s' = (0, \lambda)$.
- $L((0, \lambda)) = \{part_0, cell_\lambda\}$
- $L((i, b)) = \{part_i\} \cup \{bit \mid b = 1\}$

The propositional alphabet Σ consists of the set of propositions described above. The states of the M are either of the form $(0, \lambda)$, where λ is the content of a cell, or (i, b) for $1 \leq i \leq cn$, where b is the current bit in the encoding of the cell's position. The initial state is $(0, (q_0, \emptyset))$, indicating that a) this is the first part of the cell's encoding, b) the head is on this cell, c) the machine is in the initial state q_0, and d) the cell is blank (this should be the cell immediately to the left of the input word x).

The transition relation ensures only that the system progresses consistently from part 0 of the encoding to part 1, part 2, part 3, and so on until part cn, after which it resets back to part 0 (of the next cell). Note that the values of λ and b are unconstrained, as these will be handled by the formula φ. Observe the three consistency conditions required for runs of T are not wired into the model.

Finally, the labeling function L simply converts the state into an appropriate propositional representation.

The Formula We now construct the LTLf formula φ over the propositional alphabet Σ. As mentioned before, we want φ to be such that, if an execution of the system M has a prefix that satisfies φ, then either that execution violates a consistency condition or it is an accepting run. To achieve this, we construct $\varphi = \neg\varphi_{cons} \vee \varphi_{acc}$. φ_{acc} is defined as follows:

$$\varphi_{acc} = \bigvee_{q \in F} \bigvee_{\gamma \in \Gamma} \mathrm{F}\, cell_{(q,\gamma)}.$$

It is easy to see that an execution of M has a prefix that satisfies φ_{acc} iff that execution reaches a state $(0, (q, \gamma))$ where q is an accepting state of T.

Meanwhile, we define φ_{cons} as a conjunction of formulas, such that if an execution has a prefix that violates one of these formulas then the execution is

inconsistent, and every inconsistent execution has a prefix that violates one of these formulas. We classify these formulas into three groups, one for each of the three consistency conditions described above:

(C1). Consistency within a configuration (the binary encoding of each cell's position is correct)
(C2). Consistency with the input word (the first configuration is correct)
(C3). Consistency with the transition function (every configuration follows from the previous one)

The first two conditions (C1) and (C2) are relatively straightforward to encode as formulas of polynomial size. For details, refer to the appendix.

The third condition (C3) is where the biggest challenge lies. This condition requires reasoning about changes from one configuration to the next. The difficulty lies in accessing the segment that represents the same cell in the next configuration using a polynomial-sized formula. Recall that a cell is represented by $cn + 1$ assignments in the trace and each configuration is composed of 2^{cn} cells. Since the size of each configuration is exponential, formulas may require exponential size. For instance, if the segment representing a cell begins at assignment i in the trace, then the same cell in the next configuration will start at assignment $i + 2^{cn}(cn + 1)$. Referring to this assignment directly in the formula would require $2^{cn}(cn + 1)$ nested X operators. Alternatively, the cell in the next configuration can be identified by being the first cell where the binary encoding of its position on the tape is the same as the current cell. However, this may require enumeration on all possible assignments of the $cn + 1$ bits.

To circumvent this problem and compare corresponding cells in two different configurations using a formula of polynomial size, we take advantage of the fact that we are dealing with finite prefixes of the trace. The insight is that we can use the last position in the trace as an anchor, so that instead of having to find the cell in the next configuration with the same position encoding, we can instead look at the last cell in the trace and test if a) it is in the next configuration, and b) it has the same position encoding. Since the formula is checked for every prefix, eventually we will find a prefix where this holds. We can then check if the contents of the cells are consistent with the transition function.

We now go into details of the formula for (C3). Consistency condition (C3) says that every configuration follows from the previous one according to T's transition function δ. As mentioned before, to ensure that we get a formula of polynomial size, the formula that we construct actually expresses the following condition: for all cells c in the prefix, if the last cell c_{Last} of the prefix is in the same position as c but in the next configuration, then c_{Last} follows from c based on the transition function. Since the formula must hold for all prefixes, its satisfaction implies the original consistency condition.

We start by defining the useful shorthand $L^{-i}\phi \equiv \mathrm{F}(\phi \wedge \mathrm{X}^{i-1}\neg\mathrm{X}\,\mathsf{true})$, which denotes that ϕ holds i positions before the end of the prefix (e.g. $L^{-1}\phi$ means that ϕ holds at the last position of the prefix). This is expressed by saying that at some point in the future ϕ holds, and $i - 1$ positions after that is the last

position of the prefix (by the semantics of LTLf, $\neg \mathrm{X}\,\mathsf{true}$ only holds at the last position). We then define the formula MatchLastCell, which checks if the cell c in the current position corresponds to the last cell c_{Last} of the prefix, as follows:

$$\mathsf{MatchLastCell} \equiv \mathit{part}_0 \wedge L^{-cn}\mathit{part}_0 \wedge \bigwedge_{i=1}^{cn}(\mathrm{X}^i \mathit{bit} \leftrightarrow L^{-cn}\mathrm{X}^i \mathit{bit}) \\ \wedge\ \mathrm{X}\Big(\neg\mathsf{NewConfig}\,\mathrm{U}\,\big(\mathsf{NewConfig} \wedge \mathrm{X}\,\mathrm{G}\,\neg\mathsf{NewConfig}\big)\Big)$$

where $\mathsf{NewConfig} \equiv (\mathit{part}_0 \wedge \bigwedge_{i=1}^{cn}(\mathrm{X}^i \neg \mathit{bit}))$ denotes the start of a new configuration (a cell whose position in the tape is encoded as 0). MatchLastCell expresses that (a) we are at the start of a cell c (part_0); (b) the last cn positions of the prefix encode another cell c_{Last} ($L^{-cn}\mathit{part}_0$); (c) c and c_{Last} are in the same tape position ($\bigwedge_{i=1}^{cn}(\mathrm{X}^i \mathit{bit} \leftrightarrow L^{-cn}\mathrm{X}^i \mathit{bit})$); and (d) we start a new configuration exactly once between c and c_{Last} ($\mathrm{X}(\neg\mathsf{NewConfig}\,\mathrm{U}\,(\mathsf{NewConfig} \wedge \mathrm{X}\,\mathrm{G}\,\neg\mathsf{NewConfig}))$). In other words, c and c_{Last} are the same cell in successive configurations. We can then encode the consistency condition by the formula

$$\mathrm{G}(\mathsf{MatchLastCell} \rightarrow \varphi_\delta) \wedge \mathrm{G}(\mathsf{MatchLastCell} \rightarrow \varphi_\delta^{\leftarrow}) \\ \wedge\ \mathrm{G}(\mathrm{X}^{cn+1}\,\mathsf{MatchLastCell} \rightarrow \varphi_\delta^{\rightarrow}) \wedge \mathrm{G}(\mathrm{X}^{cn+1}\,\mathsf{MatchLastCell} \rightarrow \varphi_\delta^{0})$$

where each of φ_δ, $\varphi_\delta^{\leftarrow}$, $\varphi_\delta^{\rightarrow}$, and φ_δ^0 expresses one way in which the contents of the cell c can change (or not change) in the next configuration:

- φ_δ expresses that if the head is on c ($\mathit{cell}_{(q,\gamma)}$), then in c_{Last} the head must have moved to a different cell and written the appropriate symbol γ' given by the transition relation ($L^{-cn}\,\mathit{cell}_{\gamma'}$)
- $\varphi_\delta^{\leftarrow}$ expresses that if the head is on the cell to the *right* of c ($\mathrm{X}^{cn+1}\,\mathit{cell}_{(q,\gamma_2)}$), and the transition relation requires it to move left, then in the next configuration the head must have moved to c_{Last} ($L^{-cn}\,\mathit{cell}_{(q',\gamma_1)}$))
- $\varphi_\delta^{\rightarrow}$ expresses that if the head is on the cell to the *left* of c ($\mathit{cell}_{(q,\gamma_1)}$), and the transition relation requires it to move right, then in the next configuration the head must have moved to c_{Last} ($L^{-cn}\,\mathit{cell}_{(q',\gamma_2)}$))
- Finally, φ_δ^0 expresses that if the head is neither on c nor on the cells adjacent to it ($\mathit{cell}_{\gamma_1} \wedge \mathrm{X}^{cn+1}\,\mathit{cell}_{\gamma_2} \wedge \mathrm{X}^{2(cn+1)}\,\mathit{cell}_{\gamma_3}$), then the contents of the cell don't change ($L^{-cn}\,\mathit{cell}_{\gamma_2}$)

Note that in the latter two formulas c is the cell to the right of the current cell ($\mathrm{X}^{cn+1}\,\mathsf{MatchLastCell}$) this is necessary so that $\varphi_\delta^{\rightarrow}$ and φ_δ^0 can refer to the cell to the left of c. Formula for φ_δ, $\varphi_\delta^{\leftarrow}$, $\varphi_\delta^{\rightarrow}$, and φ_δ^0 have been presented in the appendix. The size of each formula is polynomial in the size of the transition relation of the Turing Machine.

Theorem 3 (LTLf Model Checking. Lower bound). LTLf *model checking of non-terminating systems is EXPSPACE-hard.*

Proof. Let the non-terminating system M and LTLf formula $\varphi = \neg\varphi_{cons} \vee \varphi_{acc}$ be as described above. We show that an exponential-space Turing machine T accepts an input word x iff every execution of M has a finite prefix that satisfies φ, i.e., $M \models \varphi$. Note that since T is deterministic, its execution on the input word x is unique. Therefore, there is exactly one trace π of M that simulates the execution of T on x. By construction, a trace has a finite prefix that satisfies $\neg\varphi_{cons}$ iff that trace violates one of the consistency conditions. This holds for every trace of M except π. So, because no finite prefix of π satisfies $\neg\varphi_{cons}$, M model checks if and only if π has a prefix that satisfies φ_{acc}, which means that π eventually reaches an accepting state. Since π simulates T on x, this happens if and only if T accepts x. □

5.2 Final Complexity Results

Finally, we present the complexity of model-checking non-terminating systems:

Theorem 4 (MC. Non-terminating. Complexity). LTLf *model checking of non-terminating systems is EXPSPACE-complete.*

Proof. Recall, a non-terminating system $\mathcal{M}$ satisfies an LTLf formula ϕ iff $\mathcal{L}(\mathcal{M}) \cap \mathsf{pref}(\neg\phi) = \emptyset$. A naive algorithm would explicitly construct $\mathsf{pref}(\neg\phi)$ and require doubly exponential space in the size of ϕ. Instead, the approach is to construct $\mathsf{pref}(\phi)$ on-the-fly in exponential space and simultaneously evaluate the emptiness of $\mathcal{M} \cap \mathsf{pref}(\neg\phi)$. Given all three steps in the construction of $\mathsf{pref}(\phi)$ are amenable to on-the-fly constructions, this procedure follows standard on-the-fly procedures [32]. Thus, LTLf model checking of non-terminating models is in EXPSPACE. Theorem 3 establishes the matching lower bound. □

This result is unexpected as it implies that LTLf model checking is exponentially harder than LTL model checking for non-terminating systems, contrary to the prior perception that problems in LTLf tend to be as hard if not easier than their counterparts in LTL (See Table 1).

Next, we present the complexity of model-checking terminating systems:

Theorem 5 (MC. Terminating. Complexity). LTLf *model checking of terminating systems is PSPACE-complete.*

Proof. Recall that a terminating system M satisfies an LTLf formula ϕ if every execution of M satisfies ϕ. So, $M \models \phi$ iff $\mathcal{L}(M \cap A_{\neg\phi}) = \emptyset$ where $A_{\neg\phi}$ is the NFA for $\neg\phi$. Since the NFA is exponential in the size of the LTLf formula [13], an on-the-fly algorithm for non-emptiness checking of $M \cap A_{\neg\phi}$ can be performed in PSPACE. PSPACE-hardness can be proven by a trivial reduction from LTLf *satisfiability*, which is PSPACE-complete [13]. □

For LTLf synthesis, these results imply that it is much harder to verify a non-terminating transducer than a terminating transducer. Hence, to test the correctness of an LTLf synthesis tool by verifying its output strategy, it would be better for LTLf synthesis tools to generate terminating transducers. This, to the best of our knowledge, is the *first* theoretically sound evidence to use one transducer over the other in LTLf synthesis.

6 Concluding Remarks

Motivated by the recent surge in LTLf synthesis tools that are rarely verified for result correctness, this work is the *first* to investigate the problem of LTLf model checking. Noting that LTLf synthesis can generate both terminating and non-terminating transducers, we examine LTLf model checking for both possibilities. Our central result is that LTLf model checking of non-terminating models is exponentially harder than terminating models. Their complexities are EXPSPACE-complete and PSPACE-complete, respectively. This is surprising at first as it implies that LTLf model checking is harder than LTL model checking for non-terminating models, contrary to the expectation from prior comparisons between LTLf and LTL (See Table 1). In addition to being of independent interest, our results immediately lend several broad impacts:

1. They present the first theoretical evidence for the use of terminating transducers to represent the synthesized strategies in LTLf synthesis, as it would be easier to verify the correctness of the synthesized transducer.
2. Implementations of our LTLf model checking algorithms could be deployed in large-scale competitions such as the LTLf track in SYNTCOMP 2023.
3. They invite further exploration into LTLf vs LTL, as it breaks the prior perception that problems in LTLf are *as hard if not simpler* than their LTL counterparts.

These results inspire future work in the development of practical tools for model checking and synthesis as well as the development of LTLf model checking in more complex domains such as probabilistic models or under asynchrony [3,4]. It would be interesting to see how the practical implementations compare for LTLf model checking under terminating and non-terminating semantics, even though terminating models are preferred in theory.

Acknowledgements We thank the anonymous reviewers for their valuable feedback. This work has been supported by the Engineering and Physical Sciences Research Council [grant number EP/X021513/1], NASA 80NSSC17K0162, NSF grants IIS-1527668, CCF-1704883, IIS-1830549, CNS-2016656, DoD MURI grant N00014-20-1-2787, and an award from the Maryland Procurement Office.

References

1. Baier, J.A., McIlraith, S.: Planning with temporally extended goals using heuristic search. In: ICAPS. pp. 342–345. AAAI Press (2006)
2. Bansal, S., Li, Y., Tabajara, L., Vardi, M.: Hybrid compositional reasoning for reactive synthesis from finite-horizon specifications. In: AAAI. vol. 34, pp. 9766–9774 (2020)
3. Bansal, S., Namjoshi, K.S., Sa'ar, Y.: Synthesis of coordination programs from linear temporal specifications. Proceedings of the ACM on Programming Languages (POPL) (2019)

4. Bansal, S., Namjoshi, K.S., Sa'ar, Y.: Synthesis of asynchronous reactive programs from temporal specifications. In: Computer Aided Verification: 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I 30 (2018)
5. Blum, M., Kannan, S.: Designing programs that check their work. Journal of the ACM **42**(1), 269–291 (1995)
6. Brafman, R.I., De Giacomo, G.: Planning for LTLf/LDLf goals in non-markovian fully observable nondeterministic domains. In: IJCAI. pp. 1602–1608 (2019)
7. Camacho, A., Icarte, R.T., Klassen, T.Q., Valenzano, R.A., McIlraith, S.A.: LTL and beyond: Formal languages for reward function specification in reinforcement learning. In: IJCAI. vol. 19, pp. 6065–6073 (2019)
8. De Giacomo, G., Favorito, M.: Compositional approach to translate LTLf/LDLf into deterministic finite automata. In: Proceedings of the International Conference on Automated Planning and Scheduling. vol. 31, pp. 122–130 (2021)
9. De Giacomo, G., Favorito, M., Li, J., Vardi, M.Y., Xiao, S., Zhu, S.: Ltlf synthesis as and-or graph search: Knowledge compilation at work. In: Proc. of IJCAI (2022)
10. De Giacomo, G., Iocchi, L., Favorito, M., Patrizi, F.: Foundations for restraining bolts: Reinforcement learning with LTLf/LDLf restraining specifications. In: ICAPS. vol. 29, pp. 128–136 (2019)
11. De Giacomo, G., Rubin, S.: Automata-theoretic foundations of fond planning for LTLf and LDLf goals. In: IJCAI. pp. 4729–4735 (2018)
12. De Giacomo, G., Vardi, M.: Synthesis for LTL and LDL on finite traces. In: IJCAI. pp. 1558–1564. AAAI Press (2015)
13. De Giacomo, G., Vardi, M.Y.: Linear temporal logic and linear dynamic logic on finite traces. In: IJCAI. pp. 854–860. AAAI Press (2013)
14. De Giacomo, G., Vardi, M.Y.: LTLf and LDLf synthesis under partial observability. In: IJCAI. vol. 2016, pp. 1044–1050 (2016)
15. Duret-Lutz, A., Renault, E., Colange, M., Renkin, F., Aisse, A.G., Schlehuber-Caissier, P., Medioni, T., Martin, A., Dubois, J., Gillard, C., Lauko, H.: From spot 2.0 to spot 2.10: What's new? In: Shoham, S., Vizel, Y. (eds.) Computer Aided Verification - 34th International Conference, CAV 2022, Haifa, Israel, August 7-10, 2022, Proceedings, Part II. Lecture Notes in Computer Science, vol. 13372, pp. 174–187. Springer (2022)
16. Esparza, J., Křetínskỳ, J., Sickert, S.: A unified translation of linear temporal logic to ω-automata. Journal of the ACM (JACM) **67**(6), 1–61 (2020)
17. Favorito, M.: Forward ltlf synthesis: Dpll at work. arXiv preprint arXiv:2302.13825 (2023)
18. He, K., Lahijanian, M., Kavraki, L.E., Vardi, M.Y.: Reactive synthesis for finite tasks under resource constraints. In: IROS. pp. 5326–5332. IEEE (2017)
19. Jacobs, S., Perez, G.A., Schlehuber-Caissier, P.: The temporal logic synthesis format TLSF v1.2 (2023)
20. Křetínskỳ, J., Meggendorfer, T., Sickert, S.: Owl: a library for omega-words, automata, and LTL. In: ATVA. pp. 543–550. Springer (2018)
21. Kuehlmann, A., van Eijk, C.A.: Combinational and sequential equivalence checking. Logic synthesis and Verification pp. 343–372 (2002)
22. Nicola, R.D., Vaandrager, F.W.: Action versus state based logics for transition systems. In: Guessarian, I. (ed.) Semantics of Systems of Concurrent Processes, LITP Spring School on Theoretical Computer Science, La Roche Posay, France, April 23-27, 1990, Proceedings. Lecture Notes in Computer Science, vol. 469, pp. 407–419. Springer (1990)

23. Pnueli, A.: The temporal logic of programs. In: FOCS. pp. 46–57. IEEE (1977)
24. Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: POPL. pp. 179–190 (1989)
25. Safra, S.: On the complexity of omega -automata. In: [FOCS. pp. 319–327 (1988)
26. Siegel, M., Pnueli, A., Singerman, E.: Translation validation. In: Proc. of TACAS. pp. 151–166 (1998)
27. Sistla, A.P., Clarke, E.M.: The complexity of propositional linear temporal logics. Journal of the ACM (JACM) **32**(3), 733–749 (1985)
28. Tabajara, L.M., Vardi, M.Y.: Partitioning techniques in LTLf synthesis. In: IJCAI. pp. 5599–5606. AAAI Press (2019)
29. Tabakov, D., Rozier, K., Vardi, M.Y.: Optimized temporal monitors for SystemC. Formal Methods in System Design **41**(3), 236–268 (2012)
30. Thomas, W., et al.: Automata, logics, and infinite games: a guide to current research, vol. 2500. Springer Science & Business Media (2002)
31. Vardi, M.Y.: The Büchi complementation saga. In: STACS. pp. 12–22. Springer (2007)
32. Vardi, M.Y., Wolper, P.: An automata-theoretic approach to automatic program verification. In: LICS. IEEE Computer Society (1986)
33. Vardi, M.Y., Wolper, P.: Reasoning about infinite computations. Inf. Comput. **115**(1), 1–37 (1994)
34. Wells, A.M., Lahijanian, M., Kavraki, L.E., Vardi, M.Y.: LTLf synthesis on probabilistic systems. arXiv preprint arXiv:2009.10883 (2020)
35. Wolper, P., Vardi, M.Y., Sistla, A.P.: Reasoning about infinite computation paths. In: FOCS. pp. 185–194. IEEE (1983)
36. Zhu, S., Tabajara, L.M., Li, J., Pu, G., Vardi, M.Y.: Symbolic LTLf synthesis. In: IJCAI. pp. 1362–1369. AAAI Press (2017)

Appendix

6.1 Automata over Finite and Infinite words

A (nondeterministic) *automaton* is a tuple $\mathcal{A} = (\Sigma, S, \iota, \delta, F)$ where Σ is a finite set of symbols (called an alphabet), S is a finite set of states, $\iota \in S$ is the initial state, $F \subseteq S$ is the set of accepting states, and $\delta \subseteq S \times \Sigma \times S$ is the transition relation. An automaton on *finite* words is called a *nondeterministic finite-state automaton* (NFA), while an automaton over *infinite* words is called a *nondeterministic Büchi automaton* (NBA). An NFA is said to be *deterministic* (DFA) if for each state s and letter a, $|\{s' | (s, a, s') \in \delta \text{ for some } s'\}| \leq 1$. Deterministic Büchi automata (DBAs) are defined analogously.

Let $\mathcal{A}$ be an NFA. For a finite word $w = w_0 \cdots w_n \in \Sigma^*$, a *run* of $\mathcal{A}$ over w is a finite state sequence $\rho = s_0 \dots s_{n+1} \in S^+$ such that $s_0 = \iota$ and for all $i \in \{0, \dots n\}$, $(s_i, w_i, s_{i+1}) \in \delta$ holds. A run $\rho = s_0 \dots s_{n+1}$ is an *accepting run* if $s_{n+1} \in F$. A word w is *accepted* by $\mathcal{A}$ if $\mathcal{A}$ has an accepting run over w.

Let $\mathcal{B}$ be an NBA. Similarly, a run of $\mathcal{B}$ over an infinite word $w = w_0 w_1 \cdots \in \Sigma^\omega$ is an infinite sequence $\rho = s_0 s_1 \cdots \in S^\omega$ such that $s_0 = \iota$ and for all $i \in \mathbb{N}$, $(s_i, w_i, s_{i+1}) \in \delta$. Let $inf(\rho)$ denote the set of states that occur infinitely often in run ρ. A run ρ is an *accepting run* in $\mathcal{B}$ if $inf(\rho) \cap F \neq \emptyset$. An infinite word w is accepted by $\mathcal{B}$ if $\mathcal{B}$ has an accepting run over w.

We denote by $\mathcal{L}(\mathcal{B})$ (resp. $\mathcal{L}(\mathcal{A})$) the set of all words accepted by $\mathcal{B}$ (resp. $\mathcal{A}$). It is known that NFAs/DFAs recognize exactly *regular* languages while NBAs accept exactly *ω-regular* languages. In the remainder of the paper, we denote by $w_i, i \geq 0$ the i-th element in the sequence w.

6.2 Semantics of LTLf and LTL

We first give the semantics of LTLf formulas. A finite sequence ρ over 2^{Prop} is said to satisfy an LTLf formula ϕ over Prop, denoted by $\rho \models \phi$, if $\rho, 0 \models \phi$ where for all positions $0 \leq i < |\rho|$, $\rho, i \models \phi$ is defined inductively on ϕ as follows:

- $\rho, i \models \mathsf{true}$,
- $\rho, i \not\models \mathsf{false}$,
- $\rho, i \models a$ iff $a \in \rho_i$ where ρ_i is the i-th element of ρ for all $0 \leq i < |\rho|$,
- $\rho, i \models \neg\phi$ iff $\rho, i \not\models \phi$,
- $\rho, i \models \phi_1 \wedge \phi_2$ iff $\rho, i \models \phi_1$ and $\rho, i \models \phi_2$,
- $\rho, i \models \phi_1 \vee \phi_2$ iff $\rho, i \models \phi_1$ or $\rho, i \models \phi_2$,
- $\rho, i \models \mathrm{X}\phi$ iff $i + 1 < |\rho|$ and $\rho, i + 1 \models \phi$,
- $\rho, i \models \phi_1 \mathrm{U} \phi_2$ iff there exists j s.t. $i \leq j < |\rho|$ and $\rho, j \models \phi_2$, and for all k, $i \leq k < j$, we have $\rho, k \models \phi_1$,
- $\rho, i \models \mathrm{F}\phi$ iff there exists j s.t. $i \leq j < |\rho|$ and $\rho, j \models \phi$,
- $\rho, i \models \mathrm{G}\phi$ iff for all j s.t. $i \leq j < |\rho|$, $\rho, j \models \phi$.

To obtain the semantics of LTL formulas, ρ must be an infinite sequence. Thus, the length of ρ, denoted as $|\rho|$, is ∞. It actually means that we can just

drop all restrictions that the integers need to be less than $|\rho|$ meant for LTLf semantics We use a subscript ∞ for all LTL modalities to distinguish with their LTLf counterparts. For all positions $i \geq 0$, $\rho, i \models \phi$ is defined inductively on ϕ as follows:

- $\rho, i \models \mathsf{true}$,
- $\rho, i \not\models \mathsf{false}$,
- $\rho, i \models a$ iff $a \in \rho_i$ where ρ_i is the i-th element of ρ for all $i \geq 0$,
- $\rho, i \models \neg\phi$ iff $\rho, i \not\models \phi$,
- $\rho, i \models \phi_1 \wedge \phi_2$ iff $\rho, i \models \phi_1$ and $\rho, i \models \phi_2$,
- $\rho, i \models \phi_1 \vee \phi_2$ iff $\rho, i \models \phi_1$ or $\rho, i \models \phi_2$,
- $\rho, i \models \mathrm{X}_\infty\phi$ iff $\rho, i+1 \models \phi$,
- $\rho, i \models \phi_1 \mathrm{U}_\infty \phi_2$ iff there exists j s.t. $j \geq i$ and $\rho, j \models \phi_2$, and for all k, $i \leq k < j$, we have $\rho, k \models \phi_1$,
- $\rho, i \models \mathrm{F}_\infty\phi$ iff there exists j s.t. $j \geq i$ and $\rho, j \models \phi$,
- $\rho, i \models \mathrm{G}_\infty\phi$ iff for all $j \geq i$ s.t. $j \geq i$, $\rho, j \models \phi$.

6.3 Proof of Theorem 1

Theorem 1. For LTLf formula ϕ, let $\mathsf{pref}(\phi)$ be as defined above. Then,

1. $\mathsf{pref}(\phi)$ is a safety language.
2. $\mathsf{pref}(\phi)$ is ω-regular. NBA representing $\mathsf{pref}(\phi)$ consists of $2^{2^{\mathcal{O}(|\phi|)}}$ states.

Proof of Theorem 1- 1. A language $L \subseteq \Sigma^\omega$ is a *safety language* if for every word $w \notin L$ there exists a finite-prefix u of w such for all $y \in \Sigma^\omega$ the word $u \cdot y \notin L$. Such prefixes are referred to as *bad prefix*.

Consider $w \in \Sigma^\omega$ such that $w \notin \mathsf{pref}(\phi)$. By definition of $\mathsf{pref}(\phi)$, there exists an $n > 0$ s.t. the finite-prefix $w[0, n] \models \neg\phi$. Clearly, every infinite extensions of $w[0, n]$ will also not be contained in $\mathsf{pref}(\phi)$, i.e. for all $y \in \Sigma^\omega$, $w[0, n] \cdot y \notin \mathsf{pref}(\phi)$. Hence, $\mathsf{pref}(\phi)$ is a safety language.

Proof of Theorem 1- 2. Given LTLf formula ϕ, the NBA for $\mathsf{pref}(\phi)$ can be constructed as follows:

1. Construct a DFA $D = (\Sigma, Q, \iota, \delta, F)$ for $\neg\phi$, i.e., $\mathcal{L}(D) = \mathcal{L}(\neg\phi)$.
   We require D to be *complete* in the sense that for every state s and every alphabet $a \in \Sigma$, there exists a successor $t = \delta(s, a)$.
2. Obtain a DBA $C = (\Sigma, Q, \iota, \delta', F)$ by converting all accepting states F of D to accepting sink states in C. For this, replace all outgoing transitions from all accepting states in D with self loops on all letters.
   Formally, replace every $\delta(f, a) = t$ in DFA D with $f = \delta'(f, a)$ in DBA C, for all $f \in F$ and $a \in \Sigma$. For all other states, let δ' behaves identically to δ.
3. Obtain the desired NBA $B = (\Sigma, Q, \iota, \delta', \mathcal{F} = Q \setminus F)$ by swapping accepting and non-accepting states of C.

Since C is a DBA with all accepting states as sink states, swapping accepting and non-accepting states results in its complementation. Hence, it is sufficient to show that $\mathcal{L}(C)$ accepts the complement of $\mathsf{pref}(\phi)$. In other words, C accepts $w \in \Sigma^\omega$ iff there exists a finite-prefix of w that satisfies $\neg\phi$. Clearly, $w \in \mathcal{L}(C)$ then w must have a finite-prefix satisfying $\neg\phi$ since the accepting states of C and D are identical and all but outgoing transitions from accepting states are retained. Conversely, let $w \in \Sigma^\omega$ such that it contains a finite prefix that satisfies $\neg\phi$. Despite δ and δ' being different, we need to show that w is accepted. Let v be the shortest prefix of w satisfying $\neg\phi$. Since D is a DFA, v has a unique run in D. This run also appears in C because all transitions appearing in this run in D are retained in C as none of them are outgoing transitions from accepting states (if it weren't so, then v would not have been the shortest prefix of w that satisfies $\neg\phi$). Further, since accepting states in C are sink states, $w \in \mathcal{L}(C)$. Finally, the number of states of C are bounded by those of D which is doubly exponential in $|\phi|$ [13]. □

6.4 Proof of Lemma 1

Lemma 1 Let L_n and F_n be as defined above. Then

$$L_n \uplus \{0,1,\#\}^\omega = \mathsf{pref}(F_n \uplus \{0,1,\#\}^*).$$

Proof. First, we show that $L_n \uplus \{0,1,\#\}^\omega \subseteq \mathsf{pref}(F_n \uplus \{0,1,\#\}^*)$. Trivially, all prefixes of words in $\{0,1,\#\}^\omega$ are contained in $\{0,1,\#\}^*$ since "&" does not appear in any of them. It remains to show that $L_n \subseteq \mathsf{pref}(F_n \uplus \{0,1,\#\}^*)$. Let $u \cdot \& \cdot v \in L_n$. We establish that all prefixes of $u \cdot \& \cdot v$ are contained in $F_n \uplus \{0,1,\#\}$. We perform case analysis of prefixes:

1. When the prefix is a prefix of u. These prefixes are contained in $\{0,1,\#\}^*$ since "&" does not appear in the prefix.
2. When prefix of is of the form $u \cdot \&$. Now, $u \cdot \& \in F_n$ since it contains exactly one "&" and the end of $u \cdot \&$ is not in the form $\#w\#$ for $w \in \{0,1\}^n$.
3. When prefix is of the form $u \cdot \& \cdot y$ but y does not end in $\#w\#$ for $w \in \{0,1\}^n$. For the same reason as above, $u \cdot \& \cdot y \in F_n$.
4. When prefix is of the form $u \cdot \& \cdot y$ and y ends in $\#w\#$ for $w \in \{0,1\}^n$. Since $u \cdot \& \cdot v \in L_n$, we know that every $\#w\#$ appearing in v "&" must have appeared in u, for $w \in \{0,1\}^n$. Since y is a prefix of v and $\#w\#$ is at the end of y, we get that $\#w\#$ must have also appeared in u. Hence, $u \cdot \& \cdot y \in F_n$.

Hence, $L_n \uplus \{0,1,\#\}^\omega \subseteq \mathsf{pref}(F_n \uplus \{0,1,\#\}^*)$.

Next, we prove $\mathsf{pref}(F_n \uplus \{0,1,\#\}^*) \subseteq L_n \uplus \{0,1,\#\}^\omega$. First, observe that for $x \in \mathsf{pref}(F_n \uplus \{0,1,\#\}^*)$, x can contain at most one occurrence of "&". By case analysis:

1. If x does not contain "&", then clearly, $x \in \{0,1,\#\}^\omega$.
2. Otherwise, the word is of the form $u \cdot \& \cdot v$ where $u \in \{0,1,\#\}^*$ and $v \in \{0,1,\#\}^\omega$. Either there are no occurrences of $\#w\#$ in v, for $w \in \{0,1\}^n$. In this case, $u \cdot \& \cdot v \in L_n$ vacuously.

Otherwise, there are occurrences of $\#w\#$ in v. Let $u \cdot \& \cdot y$ be an arbitrary prefix of $u \cdot \& \cdot v$ that ends in $\#w\#$. Since $u \cdot \& \cdot y \in F_n \uplus \{0,1,\#\}^*$, $u \cdot \& \cdot y \in F_n$. Thus, $\#w\#$ must have appeared in u as well. Finally, since there are only finitely many possibilities of words of the form $\#w\#$, we conclude that every occurrence of $\#w\#$ in v must have also appeared in u. Hence, $u \cdot \& \cdot v \in L_n$.

Hence, $\mathsf{pref}(F_n \uplus \{0,1,\#\}^*) \subseteq L_n \uplus \{0,1,\#\}^\omega$.

Therefore $\mathsf{pref}(L_{\psi_n}) = L_n \uplus \{0,1,\#\}^\omega$. □

6.5 Encoding of (R1) from Theorem 2

$$\begin{aligned}\mathsf{OnlyOneProp} := {} & \mathrm{G}(0 \rightarrow \neg 1 \wedge \neg \& \wedge \neg \#) \wedge \mathrm{G}(1 \rightarrow \neg 0 \wedge \neg \& \wedge \neg \#) \\ & \wedge \mathrm{G}(\& \rightarrow \neg 0 \wedge \neg 1 \wedge \neg \#) \wedge \mathrm{G}(\# \rightarrow \neg 0 \wedge \neg 1 \wedge \neg \&) \wedge \mathrm{G}(0 \vee 1 \vee \& \vee \#).\end{aligned}$$

6.6 Proof of Lemma 2

Lemma 2 Let $\phi \in \mathsf{LTLf}_{\setminus\{\mathrm{R},\vee\}}$ and let LTL $t(\phi)$ be as defined above. Then, $\mathcal{L}(t(\phi)) = \mathsf{pref}(\phi)$ and $\mathcal{O}(|\phi|) = \mathcal{O}(|t(\phi)|)$.

Proof. Trivially, $\mathcal{O}(|\phi|) = \mathcal{O}(|t(\phi|)$ holds. We prove that $\mathcal{L}(t(\phi)) = \mathsf{pref}(\phi)$ by structural induction on ϕ. Let $w = w_0 w_1 \cdots \in \Sigma^\omega$ where w_i is the i-th letter in w. Recall, $w[0,n]$ denotes the subsequence $w_0 \cdots w_{n-1}$ of w for $n > 0$. Then

- $\phi = \ell$ (resp. $\phi = \neg\ell$). By definition, $t(\phi) = \ell$. It is trivial that $w \in \mathsf{pref}(\phi) = \mathsf{pref}(\ell)$ iff $w \in \mathcal{L}(\ell)$ since either $w_0 \models \ell$ or $w_0 \not\models \ell$.
- $\phi = \psi_1 \wedge \psi_2$. Then $t(\phi) = t(\psi_1) \wedge t(\psi_2)$. Assume that $w \in \mathcal{L}(t(\psi_1) \wedge t(\psi_2))$. By LTL semantics, $w \in \mathcal{L}(t(\psi_1))$ and $w \in \mathcal{L}(t(\psi_2))$. It follows that $w \in \mathsf{pref}(\psi_1)$ and $w \in \mathsf{pref}(\psi_2)$, based on induction assumption. It means that for all $i > 0$, $w[0,i] \models \psi_1$ and $w[0,i] \models \psi_2$. Thus, $w[0,i] \models \psi_1 \wedge \psi_2$ for all $i > 0$. We then have that $w \in \mathsf{pref}(\phi)$.
  Assume that $w \in \mathsf{pref}(\phi) = \mathsf{pref}(\psi_1 \wedge \psi_2)$. It follows that for all $i > 0$, $w[0,i] \models \psi_1 \wedge \psi_2$, i.e., $w \in \mathsf{pref}(\psi_1)$ and $w \in \mathsf{pref}(\psi_2)$. By induction assumption, we have that $w \in \mathcal{L}(t(\psi_1))$ and $w \in \mathcal{L}(t(\psi_2))$. Consequently, $w \models t(\psi_1) \wedge t(\psi_2)$, i.e., $w \in \mathcal{L}(t(\psi_1) \wedge t(\psi_2))$.
- $\phi = \mathrm{F}\psi$. Then $t(\phi) = t(\psi)$. By induction assumption, we have that $w \in \mathsf{pref}(\psi)$ iff $w \in \mathcal{L}(t(\psi))$.
  Assume that $w \in \mathcal{L}(t(\phi)) = \mathcal{L}(t(\psi))$, i.e., $w \in \mathsf{pref}(\psi)$. It follows that for every $i > 0$, $w[0,i] \models \psi$. Obviously, for every $i > 0$, $w[0,i] \models \mathrm{F}\psi$. Consequently, $w \in \mathsf{pref}(\phi)$.
  Assume that $w \in \mathsf{pref}(\phi)$. Then for every $i > 0$, $w[0,i] \models \phi = \mathrm{F}\psi$. By semantics of LTLf, $w[0,1] \models \psi$, i.e., $w_0 \models \psi$. It follows that for every $i > 0$, we also have that $w[0,i] \models \psi$, indicating that $w \in \mathsf{pref}(\psi)$. By induction assumption, $w \in \mathcal{L}(t(\psi)) = \mathcal{L}(t(\phi))$. So we are done for this case.

- $\phi = \psi_1 \mathrm{U} \psi_2$. Then $t(\phi) = t(\psi_2)$. The proof is quite similar to the one for $\mathrm{F}\psi$. By induction assumption, we have that $w \in \mathsf{pref}(\psi_2)$ iff $w \in \mathcal{L}(t(\psi_2))$.
  Assume that $w \in \mathcal{L}(t(\phi)) = \mathcal{L}(t(\psi_2))$, i.e., $w \in \mathsf{pref}(\psi_2)$. It follows that for every $i > 0$, $w[0,i] \models \psi_2$. Obviously, for every $i > 0$, $w[0,i] \models \psi_1 \mathrm{U} \psi_2$ since $w[0,1] \models \psi_2$. Consequently, $w \in \mathsf{pref}(\phi)$.
  Assume that $w \in \mathsf{pref}(\phi)$. Then for every $i > 0$, $w[0,i] \models \phi = \psi_1 \mathrm{U} \psi_2$. By semantics of LTLf, $w[0,1] \models \psi_2$, i.e., $w_0 \models \psi_2$. It follows that for every $i > 0$, we also have that $w[0,i] \models \psi_2$, indicating that $w \in \mathsf{pref}(\psi_2)$. By induction assumption, $w \in \mathcal{L}(t(\psi_2)) = \mathcal{L}(t(\phi))$. So we are done for this case.
- $\phi = \mathrm{G}\psi$. Then, $t(\phi) = \mathrm{G}_\infty(t(\psi))$. By induction assumption, we have that we have that $w \in \mathsf{pref}(\psi)$ iff $w \in \mathcal{L}(t(\psi))$.
  Assume that $w \in \mathcal{L}(t(\phi)) = \mathcal{L}(\mathrm{G}_\infty(t(\psi)))$. By semantics of LTL, for every $i \geq 0, w[i,\infty] \in \mathcal{L}(t(\psi))$. In other words, we have that $w[i,\infty] \in \mathsf{pref}(\psi)$ for all $i \geq 0$. It follows that $w[i,i+1] \models \psi$ for all $i \geq 0$, according to definition of pref languages. Then we have that $w[0,i] \models \mathrm{G}\psi$ for all $i \geq 0$ in LTLf semantics. Obviously, $w \in \mathsf{pref}(\mathrm{G}\psi)$.
  Assume that $w \in \mathsf{pref}(\mathrm{G}\psi)$. By definition of pref languages, we have that $w[0,i] \models \mathrm{G}\psi$ for all $i > 0$. By semantics of LTLf, we have $w_i \models \psi$ for all $i \geq 0$ (The last position of the word needs to satisfy ψ); Also, $w[i,j] \models \psi$ for all $j > i$. By definition of pref languages, we have that $w[i,\infty] \in \mathsf{pref}(\psi)$ for all $i \geq 0$. Based on induction assumption, we have $w[i,\infty] \in \mathcal{L}(t(\psi))$ for all $i \geq 0$. It follows that $w \models \mathrm{G}_\infty t(\psi)$. Thus, we have done for this case.
- $\phi = \mathrm{N}\psi$. Then $t(\phi) = \mathrm{X}_\infty t(\psi)$. By induction assumption, $w \in \mathsf{pref}(\psi)$ iff $w \in \mathcal{L}(t(\psi))$.
  Assume that $w \in \mathsf{pref}(\phi)$. Then $w[0,i] \models \mathrm{N}\psi$ for all $i > 0$, including $i = 2$. By LTLf semantics, it follows that $w[1,2] \models \psi$ since w_0 is not the last position when $i = 2$. It follows that we have that $w[1,i] \models \psi$ for all $i > 1$. So, $w[1,\infty] \in \mathsf{pref}(\psi)$, i.e., $w[1,\infty] \in \mathcal{L}(t(\psi))$ based on induction assumption. Then we have $w \models \mathrm{X}_\infty t(\psi)$, i.e., $w \in \mathcal{L}(\mathrm{X}_\infty t(\psi))$.
  Assume that $w \in \mathcal{L}(\mathrm{X}_\infty t(\psi))$. Then $w[1,\infty] \models t(\psi)$. By induction assumption, we have $w[1,\infty] \in \mathsf{pref}(\psi)$. It follows that $w[1,2] \models \psi$ by definition of pref languages. Then $w[0,2] \models \mathrm{N}\psi$. Clearly, we have $w[0,i] \models \mathrm{N}\psi$ for all $i > 0$, including when $i = 1$. Consequently, we have $w \in \mathsf{pref}(\phi)$.
- $\phi = \mathrm{X}\psi$. Then $t(\phi) = \mathsf{false}$. It is impossible for a word $w \in \mathsf{pref}(\mathrm{X}\psi)$ to hold since $w[0,1] \not\models \mathrm{X}\psi$ as there is no next position at position 0. Therefore, $\mathcal{L}(\mathsf{pref}(\phi)) = \mathcal{L}(\mathsf{false}) = \emptyset$ since there are no words satisfying false.

□

6.7 Missing details from Section 5.1

Consistency conditions (C1) and (C2) We present the encoding of the first two consistency conditions (C1) and (C2). Recall, we require the following two:

(C1). Consistency within a configuration (the binary encoding of each cell's position is correct)
(C2). Consistency with the input word (the first configuration is correct)

Condition (C1) only needs to reason about adjacent cells in the same configuration. If $(b_1, \ldots, b_{cn})$ and $(b'_1, \ldots, b'_{cn})$ are the binary encodings of the positions of two adjacent cells, and $\mathit{Succ}(b_1, \ldots, b_{cn}, b'_1, \ldots, b'_{cn})$ is a propositional formula capturing that $(b'_1, \ldots, b'_{cn})$ encodes the successor (mod 2^{cn}) of $(b_1, \ldots, b_{cn})$ (see below for details), then the formula

$$\mathrm{G}\big((\mathit{part}_0 \wedge \mathrm{X}^{2cn+1}\, \mathit{true}) \rightarrow \mathit{Succ}(\mathrm{X}^1\, \mathit{bit}, \ldots, \mathrm{X}^{cn}\, \mathit{bit}, \mathrm{X}^{cn+2}\, \mathit{bit}, \ldots, \mathrm{X}^{2cn+1}\, \mathit{bit})\big)$$

expresses that if we start at the beginning of the encoding of a cell (part_0) and the prefix is long enough to include the entirety of the successor cell ($\mathrm{X}^{2cn+1}\, \mathsf{true}$), then *Succ* holds between the encodings of the two cells (note that b_i is given by $\mathrm{X}^i\, \mathit{bit}$ and b'_i is given by $\mathrm{X}^{cn+1+i}\, \mathit{bit}$). Similarly, the formula $\mathrm{X}^{cn}\, \mathsf{true} \rightarrow \bigwedge_{i=1}^{cn} \mathrm{X}^i \neg \mathit{bit}$ expresses that the encoding of the first cell's position is 0.

Condition (C2) only requires looking at the n cell contents that should contain the input word in the first configuration, plus ensuring that all other cells on the tape are blank. Checking the cells that should contain the input word can be expressed by the formula

$$\mathrm{X}^{(cn+1)n}\, \mathsf{true} \rightarrow \Big(\bigwedge_{i=1}^{n} \mathrm{X}^{(cn+1)i}\, \mathit{cell}_{x_i} \Big)$$

meaning that if the prefix is long enough to cover all n cells ($\mathrm{X}^{(cn+1)n}\, \mathsf{true}$), then the content of the i-th cell is x_i ($\mathrm{X}^{(cn+1)i}\, \mathit{cell}_{x_i}$), for all i from 1 to n. Ensuring that all other cells are blank can likewise be expressed by a formula of polynomial size (see below for details).

Consistency within a configuration. As explained above, the first consistency condition can be represented by a conjunction of the formula $\mathrm{X}^{cn}\, \mathsf{true} \rightarrow \bigwedge_{i=1}^{cn} \mathrm{X}^i \neg \mathit{bit}$, which expresses that the encoding of the first cell's position is 0, and the formula $\mathrm{G}((\mathit{part}_0 \wedge \mathrm{X}^{2cn+1}\, \mathsf{true}) \rightarrow \mathit{Succ}(\mathrm{X}^1\, \mathit{bit}, \ldots, \mathrm{X}^{cn}\, \mathit{bit}, \mathrm{X}^{cn+2}\, \mathit{bit}, \ldots, \mathrm{X}^{2cn+1}\, \mathit{bit}))$, which expresses that the encoded position of each successive cell is the successor of the previous one. The propositional formula *Succ* can be defined as

$$\mathit{Succ}(b_1, \ldots, b_{cn}, b'_1, \ldots, b'_{cn}) = (b'_1 \leftrightarrow \neg b_1) \wedge \bigwedge_{i=2}^{cn} (b'_i \leftrightarrow (b_i \oplus (b_{i-1} \wedge \neg b'_{i-1})))$$

which expresses the successor relation between two binary numbers $b_{cn} \ldots b_1$ and $b'_{cn} \ldots b'_1$ (note that we consider b_1 the least significant digit). The subformula $(b'_1 \leftrightarrow \neg b_1)$ expresses that the least significant digit is flipped, while $(b'_i \leftrightarrow (b_i \oplus (b_{i-1} \wedge \neg b'_{i-1})))$ (where $\oplus$ is the exclusive-or operator) expresses that the i-th digit is flipped if there is a carry (which only happens when the $(i-1)$-th digit has flipped from 1 to 0).

Consistency with the input word. The second consistency condition is composed of two formulas. As explained above, the formula $\mathrm{X}^{(cn+1)n}\, \mathsf{true} \rightarrow$

$\left(\bigwedge_{i=1}^{n} \mathrm{X}^{(cn+1)i}\, cell_{x_i}\right)$ expresses that the first n cells of the first configuration contain the input word $x = x_1 \dots x_n$. The second formula ensures that all other cells are blank, and can be expressed by

$$\mathrm{X}^{(cn+1)(n+1)}\, \mathsf{true} \rightarrow \mathrm{X}^{(cn+1)(n+1)} \Big((part_0 \rightarrow cell_\emptyset)\, \mathrm{W}\, \big(part_0 \wedge \bigwedge_{i=1}^{cn} \mathrm{X}^i \neg bit\big)\Big)$$

meaning that if the prefix is long enough to reach the $(n+1)$-th cell ($\mathrm{X}^{(cn+1)(n+1)}\, \mathsf{true}$), the contents of every cell from this point on must be blank ($part_0 \rightarrow cell_\emptyset$) until we reach a new configuration, indicated by the encoding of the cell position resetting back to 0 ($part_0 \wedge \bigwedge_{i=1}^{cn} \mathrm{X}^i \neg bit$). Note that the "zeroth" cell (the cell where the head starts, immediately before the input word) is also blank, but this is enforced by the transition relation of M.

Missing formulas for (C3)

- φ_δ expresses that if the head is on c ($cell_{(q,\gamma)}$), then in c_{Last} the head must have moved to a different cell and written the appropriate symbol γ' given by the transition relation ($L^{-cn}\, cell_{\gamma'}$)

$$\varphi_\delta \equiv \bigwedge_{q \in Q} \bigwedge_{\gamma \in \Gamma} \big(cell_{(q,\gamma)} \rightarrow L^{-cn}\, cell_{\gamma'}\big) \qquad \text{where } \delta(q,\gamma) = (q', \gamma', d)$$

- $\varphi_\delta^{\leftarrow}$ expresses that if the head is on the cell to the *right* of c ($\mathrm{X}^{cn+1}\, cell_{(q,\gamma_2)}$), and the transition relation requires it to move left, then in the next configuration the head must have moved to c_{Last} ($L^{-cn}\, cell_{(q',\gamma_1)}$))

$$\varphi_\delta^{\leftarrow} \equiv \bigwedge_{q \in Q} \bigwedge_{\gamma_1 \in \Gamma} \bigwedge_{\gamma_2 \in \Gamma} \Big(\big(cell_{\gamma_1} \wedge \mathrm{X}^{cn+1}\, cell_{(q,\gamma_2)}\big) \rightarrow L^{-cn}\, cell_{(q',\gamma_1)}\Big)$$

where $\delta(q,\gamma_2) = (q', \gamma', \leftarrow)$

- $\varphi_\delta^{\rightarrow}$ expresses that if the head is on the cell to the *left* of c ($cell_{(q,\gamma_1)}$), and the transition relation requires it to move right, then in the next configuration the head must have moved to c_{Last} ($L^{-cn}\, cell_{(q',\gamma_2)}$))

$$\varphi_\delta^{\rightarrow} \equiv \bigwedge_{q \in Q} \bigwedge_{\gamma_1 \in \Gamma} \bigwedge_{\gamma_2 \in \Gamma} \Big(\big(cell_{(q,\gamma_1)} \wedge \mathrm{X}^{cn+1}\, cell_{\gamma_2}\big) \rightarrow L^{-cn}\, cell_{(q',\gamma_2)}\Big)$$

where $\delta(q,\gamma_1) = (q', \gamma', \rightarrow)$

- Finally, φ_δ^0 expresses that if the head is neither on c nor on the cells adjacent to it ($cell_{\gamma_1} \wedge \mathrm{X}^{cn+1}\, cell_{\gamma_2} \wedge \mathrm{X}^{2(cn+1)}\, cell_{\gamma_3}$), then the contents of the cell don't change ($L^{-cn}\, cell_{\gamma_2}$)

$$\varphi_\delta^0 \equiv \bigwedge_{\gamma_1 \in \Gamma} \bigwedge_{\gamma_2 \in \Gamma} \bigwedge_{\gamma_3 \in \Gamma} \Big(\big(cell_{\gamma_1} \wedge \mathrm{X}^{cn+1}\, cell_{\gamma_2} \wedge \mathrm{X}^{2(cn+1)}\, cell_{\gamma_3}\big) \rightarrow L^{-cn}\, cell_{\gamma_2}\Big)$$

Note that in the latter two formulas c is the cell to the right of the current cell ($\mathrm{X}^{cn+1}\, \mathsf{MatchLastCell}$) this is necessary so that $\varphi_\delta^{\rightarrow}$ and φ_δ^0 can refer to the cell to the left of c.

6.8 NBA with at least 2^{2^n} states

Let $n \in \mathbb{N}$ and $\Sigma = \{0, 1, \#, \&\}$. Consider the language $L_n \subseteq \Sigma^\omega$ where

$$u \cdot \& \cdot v \in L_n \text{ s.t. if } \#w\# \text{ appears in } u \text{ then } \#w\# \text{ also appears in } v,$$

where $w \in \{0,1\}^n$, $u \in \{0,1,\#\}^*$ and $v \in \{0,1,\#\}^\omega$. Essentially, L_n is a bit-level adaption of the language K_d where $x \cdot \& \cdot y \in K_D$ if digits appearing in x are a subset of digits appearing in y, where $x \in D^*$ and $y \in D^\omega$ for $D = \{0, 1, \cdots, d-1\}$. We show that all NBA of K_d consists of at least 2^D states. This proof can easily be adapted to show that all NBA of L_n consists of $2^{2^{\Omega(n)}}$ states.

First, note that K_d is a safety ω-regular language. Let C_d be an (non-deterministic Büchi) automaton representing K_d. Then, C_d can be trimmed by removing all states that are unreachable from the initial state and at least one accepting state. Next, all states of the trimmed automaton can be converted to accepting states. Let us denote this automaton by A_d. Clearly, $L(A_d) = L(C_d)$ and A_d has fewer states than C_d.

We claim that C_d must have at least 2^d states. Suppose there are fewer than 2^d states. We will use the notation x_S and y_T to denote finite and infinite words over the digits D s.t. S and T denote the set of digits appearing in x_S and y_T respectively. For a state Q in A_d with outgoing transitions on &, let $\& \cdot y_{T_1}, \ldots \& \cdot y_{T_p}$ be all the infinite words with paths starting in Q. Since all paths are accepting (A_d is a safety automaton), all finite words to Q must be of the form x_S where $S \subseteq T$ and $T = \bigcap_{i=1}^{p} T_i$. Now, consider a word $x_T \& y_T$. We claim that all its accepting paths must pass through states of the form Q. Suppose x_T has a path to a state Q' with an outgoing transition on &. Similar to T for Q, let T' be defined for Q'. We assume $T' \neq T$. Clearly, $T \subseteq T'$, since otherwise it would accept a word $x_T \& y_S$ where $T \nsubseteq S$. Furthermore, $T' \subseteq T$ since otherwise $\& \cdot y_T$ will not have a path from Q'. Hence, $T = T'$. Hence, for every $S \subseteq D$, A_d must have at least one unique state to accept words of the form $x_S \& y_S$. Thus, A_d must have at least 2^D states. Subsequently, all automata C_d of the language must contain at least 2^D states.