LTL can be more succinct

Kamal Lodaya and A V Sreejith

The Institute of Mathematical Sciences
Chennai 600113, India

Abstract. It is well known that modelchecking and satisfiability of Linear Temporal Logic (LTL) are PSPACE-complete. Wolper showed that with grammar operators, this result can be extended to increase the expressiveness of the logic to all regular languages. Other ways of extending the expressiveness of LTL using modular and group modalities have been explored by Baziramwabo, McKenzie and Thérien, which are expressively complete for regular languages recognized by solvable monoids and for all regular languages, respectively. In all the papers mentioned, the numeric constants used in the modalities are in unary notation. We show that in some cases (such as the modular and symmetric group modalities) we can use numeric constants in binary notation, and still maintain the PSPACE upper bound. Adding modulo counting to LTL[F] (with just the unary future modality) already makes the logic PSPACE-hard. We also consider a restricted logic which allows only the modulo counting of length from the beginning of the word. Its satisfiability is Σ_3^P-complete.

1 Introduction

In this theoretical paper, we consider the extension of LTL to count the number of times a proposition holds modulo n. (More generally, in a recursive syntax, we can count formulas which themselves can have counting subformulas.)

There are many such extensions: Wolper used operators based on right-linear grammars [17], Emerson and Clarke developed the μ-calculus [2]. Henriksen and Thiagarajan's dynamic LTL [8] is an extension based on ideas from process logic [6]. Harel and Sherman had used operators based on automata for PDL [7]. Another extension has propositional quantification [4], but its model checking complexity is nonelementary [18]. More recently we have PSL/Sugar, and Vardi narrates [15] how regular expressions proved to be more successful than finite automata as far as designers in industry were concerned. Baziramwabo et al [1] explicitly have countably many MOD_n^k operators for their logic LTL+MOD.

Wolper's grammars, Harel and Sherman's automata, Henriksen and Thiagarajan's regular expressions, all use in effect a unary notation to express n. Hence stating properties using a large n is cumbersome. Think of a model describing properties of a circuit (which works very fast) interleaved with events which take place at regular intervals of time, which can be thought of as happening over very long stretches of the model. A look at the model checking procedure developed by Serre for LTL+MOD [12] shows that only an EXPSPACE complexity result holds if we use binary notation.

Our first main theorem is that the PSPACE upper bound holds even when we use binary notation to represent the modulo counting, and this can be carried all the way to a logic LTL+SYM, derived from Baziramwabo et al [1], which generalizes LTL+MOD to computation in the symmetric groups S_n. Unlike Serre [12], we do not use alternating automata but ordinary NFA and the standard "formula automaton" construction in our decision procedure. We have next a technical result showing that the logic LTL[F]+MOD is already PSPACE-hard. Since LTL[F] is NP-complete, this shows that modulo counting is powerful.

So we look to weakening the modulo counting. This is done by only allowing the counting of lengths (rather than the number of times a formula holds). We show that the satisfiability problem of this logic, which we call LTL[F]+LEN, is exactly at Σ_3^P, the third level of the polynomial-time hierarchy, again irrespective of whether we use unary or binary notation.

The word "succinct" in the title of our paper is used in this simple programming sense of being able to use exponentially succinct notation. There are more sophisticated ways in which succinctness appears in temporal logics, which we do not address.

We do not know if our work will make any impact on verification [2, 13, 16], since practitioners already know that a binary counter is an inexpensive addition to a modelchecking procedure. We think the finer analysis is of some theoretical interest. We would like to see if the group operators of Baziramwabo et al [1] can be put to use in realistic examples.

We recently learnt of Laroussinie, Meyer and Petonnet's work [10] on counting CTL. We only consider linear-time logic here. Also we only consider modulo counting properties, which remain within the framework of regular languages.

Acknowledgment: We would like to thank N.R. Aravind for suggesting a simplification of the proof of Theorem 6.

2 Counting and group extensions of LTL

2.1 Modulo counting

We begin by extending the LTL syntax with modulo counting and its specialization to length counting. Its generalization to computation in an arbitrary symmetric group following Baziramwabo, McKenzie and Thérien [1] is described in the next subsection.

$\delta ::= \#\alpha \mid \delta_1 + \delta_2 \mid \delta_1 - \delta_2 \mid c\delta,\ c \in \mathbb{N}$

$\phi ::= \delta \equiv r \mod q,\ q \in \mathbb{N},\ q \geq 2,\ 0 \leq r < q$

$\alpha ::= p \in Prop \mid \phi \mid \neg\ \alpha \mid \alpha\ \vee\ \beta \mid \mathsf{X}\alpha \mid \alpha\ \mathsf{U}\ \beta$

As usual $\mathsf{F}\alpha$ abbreviates $true\mathsf{U}\alpha$ and $\mathsf{G}\alpha$ is $\neg\mathsf{F}\neg\alpha$. We will use the "length" ℓ to abbreviate $\#true$.

We denote by LTL+MOD the logic whose syntax we defined above. LTL[F]+MOD is a restriction where the U modality is not allowed. We also use notation such as LTL[F]+MOD(q) when the counting is restricted to the modulo divisor q. At times we may need to distinguish between the syntax where r and q above

are given in binary ($LTL[\mathsf{X},\mathsf{U}]{+}MOD^{bin}$) and when they are given in unary ($LTL[\mathsf{X},\mathsf{U}]{+}MOD^{un}$).

By further restricting the subterm $\#\alpha$ in the δ terms to be ℓ only, we get the logic LTL[F]+LEN which can only count lengths rather than occurrences of propositions or formulae.

We denote by PROP+LEN the logic obtained by removing even the F modality from the syntax of LTL[F]+LEN, so we have propositional logic (interpreted over a word) with some length counting operations.

Finally we denote by LTL+THRESH the logic obtained by adding $\delta < r, \delta = r, \delta > r$ to ϕ.

The semantics for LTL is given by a finite state sequence (or word) M over the alphabet $\wp(Prop)$. Our results also hold for the usual semantics over infinite words, but some of the examples are more sensible with finite words, so we will stick to that in the paper and point out how the arguments need to be changed for infinite words.

$M, i \models p$ iff $p \in M(i)$

$M, i \models \mathsf{X}\alpha$ iff $M, i+1 \models \alpha$

$M, i \models \alpha \ \mathsf{U} \ \beta$ iff for some $m \geq i : M, m \models \beta$
and for all $i \leq l < m : M, l \models \alpha$

For the counting terms, the interpretation of $\#\alpha$ at the index i in the word M is given by the cardinality of the set $\{1 \leq l \leq i \mid M, l \models \alpha\}$. The arithmetic operations in the syntax of δ are then defined.

$M, i \models \delta \equiv r \mod q$ iff the cardinality associated with δ at i in M leaves a remainder r when divided by q.

Even length words can be expressed in LTL[F]+LEN by $\mathsf{FG}(\ell \equiv 0 \mod 2)$. On the other hand an even number of occurrences of the holding of a proposition p requires an LTL[F]+MOD formula: $\mathsf{FG}(\#p \equiv 0 \mod 2)$.

We count from the beginning of the word upto and including the present point where the formula is being evaluated. It is an exercise to generalize the syntax to a modality of the form $\phi \mathsf{S} \beta$, which would count from a past occurrence of β (but which one?). Similarly we could introduce counting in the future by $\phi \mathsf{U} \beta$ which would count from the present until a future occurrence of β (but which one?). Supposing we needed the number of occurrences of the proposition p before we hit β to be divisible by 3, we could write this using a disjunction of three possibilities, where the present count of p is $i \mod 3$ and that at β is also $i \mod 3$. We are assured by Baziramwabo et al [1] that LTL[X,U]+MOD is expressively complete for the logic FO+MOD [14], so we stick to their simple syntax.

2.2 Group extension

Now we follow Baziramwabo, McKenzie and Thérien [1] to generalize the modulo counting to a kind of computation in symmetric groups. Our syntax above is extended to allow

$\phi ::= \#_G(\alpha_1, \ldots, \alpha_k) = h, \ h \in G$

For the semantics, let us define $\Gamma(M,l) = g_j$ if $M,l \models \neg\alpha_1 \wedge \ldots \neg\alpha_{j-1} \wedge \alpha_j$ for $1 \leq j \leq k$. Also define $\Gamma(M,l) = 1$ (the identity element) if none of the formulae $\alpha_1, \ldots, \alpha_k$ hold at position l. Then:

$M,i \models \#_G(\alpha_1, \ldots, \alpha_k) = h$ iff $(\Pi_{l=0}^{i}\Gamma(M,l)) = h$

This generalizes the modulo counting we were doing earlier, which can be thought of as working with cyclic groups.

The groups G used in the formulae are symmetric groups specified by their generators. This extension is called LTL+SYM.

For instance, we could specify the symmetric group S_5 (shown in Figure 1) using a syntax such as

group S5(5) generators (2 3 4 5 1), (2 1 3 4 5)

which specifies a permutation group named S5 with two generators defined as permutations of the elements $(1,2,3,4,5)$ mapping these elements to the values shown. In general we define a group named G with permutations over the set $\{1,\ldots,n\}, n \geq 2$ and generators $g_1, \ldots, g_k$. Any group can be embedded in a symmetric group [9], but while using symmetric groups the group operations are implicit.

Notice that h in the syntax above is a group element, not necessarily a generator of the group. As with modulo counting, we can have a more succinct syntax by representing h using binary notation. Using the generators is also a succinct way of representing groups (see below for a standard argument). For instance, the symmetric group S_n has $n!$ elements, but can be generated by 2 generators(as shown in example) each generator being a permutation on n elements. The analogue while doing modulo counting is to use binary notation to specify the numbers r and q.

Proposition 1. *Any group has a generating set of logarithmic size.*

Proof. Let G be a group. For an $H \subseteq G$, we denote by $\langle H \rangle$ the group generated by the elements H. Take an element $g_0 \in G$. Let $H_0 = \{g_0\}$. If $\langle H_0 \rangle \neq G$, take g_1 from $G \backslash \langle H_0 \rangle$, and call $H_0 \cup \{g_1\}$ as H_1. Continue doing this until you find an H_k such that $\langle H_k \rangle = G$. We prove that $\forall i \leq k : \ |\langle H_{i+1} \rangle| \geq 2 \times |\langle H_i \rangle|$. Observe that since $g_{i+1} \notin \langle H_i \rangle$, it implies $g_{i+1}\langle H_i \rangle \cap \langle H_i \rangle = \phi$. Also $|g_{i+1}.\langle H_i \rangle| = |\langle H_i \rangle|$. But $g_{i+1}.\langle H_i \rangle \cup \langle H_i \rangle \subseteq H_{i+1}$. Therefore $|\langle H_{i+1} \rangle| \geq 2 \times |\langle H_i \rangle|$. Hence $\langle H_{log|G|} \rangle = G$. □

The picture below shows the symmetric group S_n (for $n = 5$) as the transition structure of an automaton. The language accepted can be defined by the formula $F\ (\mathsf{X}false \wedge \#_{S5}(a,b) = (12\ldots n))$ where the specification of S5 with generators was shown earlier.

3 Succinctness comes easy

Our first main theorem shows that the upper bound for LTL satisfiability can be extended to include the modulo and group counting computations, even when specified in binary.

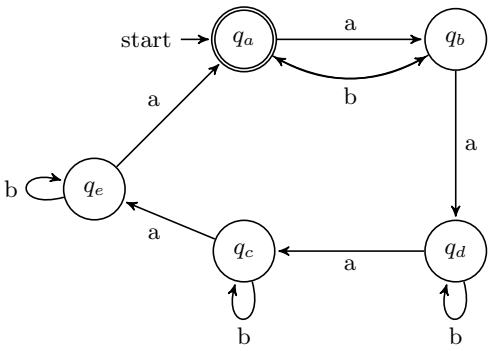

Fig. 1. An automaton representing the symmetric group S_5

Theorem 1. *If an $LTL[\mathsf{X},\mathsf{U}]+SYM+THRESH$ formula α_0 is satisfiable then there exists a satisfying model of size exponential in α_0 (even using binary notation for the formula).*

Proof. The Fischer-Ladner closure of a formula α_0 [5] is constructed as usual, where we add the following clauses:

1. The closure of $\#\alpha \equiv r \mod q$ includes α and also has $\#\alpha \equiv s \mod q$ for every s from 0 to $q-1$. (Notice that only one of these can be true at a state.)
2. The closure of $\#\alpha > q$ includes α and also has $\#\alpha = r$ for every $r \leq q$.
3. The closure of $\#_G(\alpha_1, \ldots, \alpha_k) = h$ includes $\alpha_1, \ldots, \alpha_k$ and also contains the formulae $\#_G(\alpha_1, \ldots, \alpha_k) = h'$ for every element h' of the group. (Only one of these can be true at a state.)

Observe that with binary notation, the closure of a formula α_0 can be exponential in the size of α_0, unlike the usual linear size for LTL, since the constants r, q and h are written in binary notation. A state of the tableau or formula automaton which we will construct is a maximal consistent subset of formulae from the closure of α_0. However, only one of the potentially exponentially many formulae of the form $\#\alpha \equiv s \mod q$, $0 \leq s < q$, or of the form $\#\alpha = r$, $0 \leq r \leq q$, and $\#\alpha > q$ or of the form $\#_G(\alpha_1, \ldots, \alpha_k) = h$, $h \in G$, can consistently hold. So a state is also exponential in the size of α_0. Here is a formal argument, using induction on structure of α, that the set of states of the formula automaton M_α is $2^{O(|\alpha|)}$. We denote by $|q|$ and $|G|$ for the input size (binary notation) and by S_α the number of states in M_α.

1. $\alpha = p \in P$. This is trivial.
2. $\alpha = \beta \vee \gamma$. $S_\alpha = S_\beta \times S_\gamma \leq 2^{O(|\beta|+|\gamma|)}$ (By IH)
3. $\alpha = \neg\beta$. This is just change of final states in M_β.
4. $\alpha = \beta \mathsf{U} \gamma$. $S_\alpha = S_\beta \times S_\gamma \leq 2^{O(|\beta|+|\gamma|)}$
5. $\alpha = \#\beta \equiv r \mod q$. Since any atom can have only one formula of this kind, $S_\alpha = S_\beta \times q \leq 2^{O(|\beta|+|q|)}$

6. $\alpha = \#\beta = r$. Since any atom can have only one formula of this kind, $S_\alpha = S_\beta \times q \leq 2^{O(|\beta|+|q|)}$
7. $\alpha = \#_G(\alpha_1, \ldots, \alpha_k) = h$. Again any atom can have only one formula of this kind, $S_\alpha = S_{\alpha_1} \times \cdots \times S_{\alpha_k} \times card(G) \leq 2^{O(|\alpha_1|+\cdots+|\alpha_k|+|G|)}$. □

Corollary 1. *$LTL[\mathsf{X}, \mathsf{U}]+SYM+THRESH$ satisfiability is in* PSPACE*, even using binary notation for the syntax.*

Proof. Since the formula automaton has exponentially many states, each state as well as the transition relation can be represented in polynomial space. By using moduli in binary and group generators, a state can be updated along a transition relation in polynomial time. Now we can guess and verify an accepting path in PSPACE. □

We note that this argument is not affected by whether we consider finite or infinite word models.

We now look at the model checking problem for the above logics.

Theorem 2. *1. The model complexity of the verification problem is complete for* NLOGSPACE.
2. The specification complexity of the verification problem is complete for PSPACE.

Proof. Let α_0 be a formula in $LTL[\mathsf{X}, \mathsf{U}]+SYM$ and M a Kripke structure. Theorem 1 shows that for a formula $\neg\alpha_0$ there is an exponential size model $M_{\neg\alpha_0}$ which captures the language defined by $\neg\alpha_0$. Verifying $M \vDash \alpha_0$ is equivalent to checking whether the intersection of the above automatas is non-empty. This can be done by a non-deterministic machine which uses space logarithmic in the size of both the models. Since $M_{\neg\alpha_0}$ is exponentially larger than α_0 we get the specification complexity to PSPACE. □

3.1 But modulo counting is hard

Next we consider the logic LTL[F]+MOD. It can express properties which can be expressed by LTL but not by LTL[F], for example $\mathsf{G}(p \iff \ell \equiv 1 \mod 2)$ expresses alternating occurrences of p and $\neg p$. Our next result shows that the satisfiability problem for LTL[F]+MOD, even with unary notation, is PSPACE-hard.

Theorem 3. *The satisfiability problem for $LTL[\mathsf{F}]+MOD(2)$ is* PSPACE*-hard, even with the modulo formulae restricted to counting propositions, and using unary notation for the formulae.*

Proof. Since the satisfiability problem for LTL[X,F] is PSPACE-hard [13], it is sufficient to give a polynomial-sized translation of the modality $\mathsf{X}\alpha$. This is done by introducing two new propositions p^E_α and p^O_α for each such formula, and enforcing the constraints below. Let *EvenPos* abbreviate $\ell \equiv 0 \mod 2$ and *OddPos* abbreviate $\ell \equiv 1 \mod 2$.

$$\mathsf{G}(\alpha \iff ((EvenPos \supset p^E_\alpha) \wedge (OddPos \supset p^O_\alpha)))$$

$$\mathsf{G}((EvenPos \supset \#p_\alpha^E \equiv 0 \mod 2) \wedge (OddPos \supset \#p_\alpha^O \equiv 0 \mod 2))$$

Consider p_α^E. Its count has to be an even number at every even position. Since the count increases by one if even positions satisfy α, it has to increase by one at the preceding odd position. So at an *odd* position, $\mathsf{X}\alpha$ holds precisely when the count of p_α^E is odd. Symmetrically, at an *even* position, $\mathsf{X}\alpha$ holds precisely when the count of p_α^O is odd. So we can replace an occurrence of $\mathsf{X}\alpha$ by the formula

$$(EvenPos \supset \#p_\alpha^O \equiv 1 \mod 2) \wedge (OddPos \supset \#p_\alpha^E \equiv 1 \mod 2).$$

Since α is used only once in the translation, this gives a blowup of the occurrence of $\mathsf{X}\alpha$ by a constant factor. With one such translation for every X modality, the reduction is linear. □

4 Length modulo counting

We now consider the weaker counting formulae $\ell \equiv r \mod q$, where ℓ abbreviates $\#true$. So we can only count lengths rather than propositions, which was something we needed in the PSPACE-hardness proof in the previous section.

Note that the language of alternating propositions p and $\neg p$ is in LTL[F]+LEN. It is known [13, 3, 11] that a satisfiable formula in LTL[F] has a polynomial sized model. Unfortunately LTL[F]+LEN does not satisfy a polynomial model property. Let p_i be distinct primes (in unary notation) in the following formula:

$$\mathsf{F}((\ell \equiv 0 \mod p_1) \wedge (\ell \equiv 0 \mod p_2) \wedge \cdots \wedge (\ell \equiv 0 \mod p_n)).$$

Any model which satisfies this formula will be of length at least the product of the primes, which is $\geq 2^n$. We show that the satisfiability problem of LTL[F]+LEN is in Σ_3^P, the third level of the polynomial-time hierarchy.

We give a couple of technical lemmas concerning the logic PROP+LEN which will be crucial to our arguments later.

Lemma 1. *Let α be a PROP+LEN formula. Then the following are equivalent.*

1. $(\forall w, |w| = n \implies \exists k \leq n : w, k \models \alpha)$
2. $(\exists k \leq n, \forall w : |w| = n \implies w, k \models \alpha)$

Proof. $(2 \implies 1)$: This is trivial.
$(1 \implies 2)$: Assume that the hypothesis is true but the claim is false. Let $S = \{w \mid |w| = n\}$. Pick a $w \in S$. By the hypothesis $\exists i \leq n : (w, i) \vDash \alpha$ and we can assume that there exists some $w' \in S$ such that $(w', i) \nvDash \alpha$. If this is not true then we have a witness i, such that $\forall w \in S : (w, i) \vDash \alpha$. Let u_i be the state at the i^{th} location of w'. Replace the i^{th} state in w by u_i without changing any other state in w. Call this new word w''. Now $(w'', i) \nvDash \alpha$. Again by the hypothesis, $\exists j \leq n : (w'', j) \vDash \alpha$. By the same argument given above, $\exists w''' : (w''', j) \nvDash \alpha$. We can replace the j^{th} state of w'' by the the j^{th} state from w''' which makes the resultant word not satisfy α at the j^{th} location. We can continue doing the above procedure. Since n is finite after some finite occurence of the above procedure, we will get a word v such that $\forall k \leq n : (v, k) \nvDash \alpha$. But this implies the hypothesis is wrong and hence a contradiction. □

Our next result is the following. Given a PROP+LEN formula α and two numbers m, n in binary, the problem $BlockSAT$ is to check whether there exists a model M of size $m+n$ such that $M, m \models \mathsf{G}\alpha$.

Lemma 2. *$BlockSAT$ can be checked in Π_2^P.*

Proof. The algorithm takes as input a PROP+LEN formula α, along with two numbers m, n in binary. Observe that since n is in binary we cannot guess the entire model. The algorithm needs to check whether there exists a model w satisfying α at all points between m and $m+n$, in other words, whether $\exists w : \forall k : m \leq k \leq m+n, |w| = n \wedge w, k \models \alpha$. Take the complement of this statement, which is $\forall w, |w| = n \implies \exists k : m \leq k \leq m+1, w, k \models \neg\alpha$. By the previous Lemma 1 we can check this condition by a Σ_2^P machine. Hence $BlockSAT$ can be verified by a Π_2^P machine. □

4.1 Succinct length modulo counting can be easier

We show that satisfiability of LTL[F]+LEN can be checked in Σ_3^P, even in binary notation, showing that this restriction does buy us something.

Before proceeding into an algorithm, we need to introduce a few definitions. Let α be a formula over a set of propositions P, $SubF(\alpha)$ its set of future subformulae. $prd(\alpha)$ the product over all elements of the set $\{n \mid \delta \equiv r \mod n$ is a subformula of $\alpha\}$.

Let M be a model. We define *witness index* in M for α as $\{max\{j \mid M, j \models \mathsf{F}\beta\} \mid F\beta \in SubF(\alpha)$ and $\exists i : M, i \models \beta\}$. A state at a witness index is called a *witness state*. We say $F\beta$ is witnessed at i if $i = max\{j \mid M, j \models \mathsf{F}\beta\}$. Call all states other than witness states of M as *pad states* of M for α.

We define a model M to be *normal* for α if between any two witness states of M (for α) there are at most $prd(\alpha)$ number of pad states. We claim that if α is satisfiable then it is satisfiable in a normal model.

A normal model of α will be of size $\leq |SubF(\alpha)| \times prd(\alpha)$, which is of size exponential in α. So guessing the normal model is too expensive, but we can guess the witness states (the indices and propositions true at these states), which are polynomial, verify whether the F requirements are satisfied there, and verify if there are enough pad states to fill the gap between the witness states. We will argue that we can use a Π_2 oracle to verify the latter part. The proof is given below.

Theorem 4. *Satisfiability of $LTL[\mathsf{F}]+LEN$ can be checked in Σ_3^P, even if the syntax uses binary notation.*

Proof. Let α be satisfiable. We guess the following and use it to verify whether there exists a normal word satisfying these guesses.

1. Guess k indices (positions), $u_1 < u_2 < ... < u_k$, where $k \leq |SubF(\alpha)|$ and $\forall i, u_i \leq prd(\alpha)$.
2. Guess the propositions true in the states at these k indices.

3. Guess the propositions true at the start state (if already not guessed).
4. For each of the k indices guess the set of $\mathsf{F}\beta \in SubF(\alpha)$ which are witnessed there. Let the conjunction of all formulae witnessed at u_j be called β_j. (Certain future formulae need not be true in any state in the word.)

We need to verify that there exists a word model M which is normal for α and which satisfies the guesses. Observe that the positions $1, u_1+1, ..., u_{k-1}+1$ in M should all satisfy certain G requirements (the model starts from index 1). If we have guessed that a future formula $\mathsf{F}\beta_0$ is not satisfied in the model, then the entire word should also satisfy its negation $\mathsf{G}\neg\beta_0$. Similary at state u_i+1, $\mathsf{G}\wedge_{j=0}^{i}\neg\beta_j$ should be true.

To verify that all the F, G requirements are satisfied at the witness states (the u_i indices we guessed), we start verifying from the last state u_k. All modalities can be stripped away and verified against the propositions true at this state and the location of the state. To verify $\mathsf{F}\beta_i$ at an intermediate state, We know that only those beyond the current index have been verified in future witness states. We reduce the verification of the rest to that of a pure PROP+LEN formula by making passes from the innermost subformulae outward, which can be done in polynomial time. A more formal description of this algorithm would need to keep track of the formulae satisfied and not satisfied in the future at every witness state.

To verify that the pad states between two witness states satisfy the current set of $\mathsf{G}\beta$ requirements, we need to check that the pad states should satisfy their conjunction $\bigwedge\beta$. Stripping modalities which have been verified, this is a pure PROP+LEN formula γ. What we now need to verify is that at position u_i+1, we want a word of length $u_{i+1}-u_i-1$ which satisfies $\mathsf{G}\gamma$. From Lemma 2, we see that this is the $BlockSAT$ property, checkable in Π_2^P. The algorithm we have described is an NP procedure which uses a Π_2^P oracle and hence is in Σ_3^P. □

This algorithm needs to be somewhat modified when considering satisfiability for infinite word models. First of all, we observe that we can restrict ourselves to considering "lasso" models where we have a finite prefix followed by an infinite loop, and for convenience in dealing with modulo counts, we can take the length of the loop body to be a multiple of $prd(\alpha)$. The procedure described above essentially works for the prefix part of the model, but we have to devise a further procedure which handles the requirements in the loop part of the model. Since the key to this procedure is the verification of $BlockSAT$, which remains unchanged, the extended procedure for satisfiability over infinite word models can also be carried out in Σ_3^P and Theorem 4 continues to hold.

We now look at the model checking problem.

Theorem 5. *The model checking problem for $LTL[\mathsf{F}]+LEN$ is in Σ_3^P.*

Proof. Let α_0 be a formula in $LTL[\mathsf{F}]+LEN$ and M a Kripke structure. Then one can construct a formula ϕ_M which defines the language accepted by M in polynomial time. Checking whether $M \vDash \phi$ is equivalent to checking whether $\phi_M \wedge \neg\alpha_0$ is satisfiable or not, which is in Σ_3^P. □

4.2 But length modulo counting is harder than future-LTL

In this section we show that the satisfiability problem for $LTL[\mathsf{F}]{+}LEN$ is Σ_3^P-hard, even if we use unary notation and finite word models. We denote by $\beta[\phi/p]$ the formula got by replacing all occurences of the proposition p by ϕ.

Let QBF_3 be the set of all quantified boolean formulae which starts with an existential block of quantifiers followed by a universal block of quantifiers which are then followed by an existential block of quantifiers. Checking whether a QBF_3 formula is true is Σ_3^P-complete. We reduce from evaluation of QBF_3 formulae to satisfiability of our logic.

Theorem 6. *Satisfiability for $LTL[\mathsf{F}]{+}LEN$ is hard for Σ_3^P, even if unary notation is used for the syntax.*

Proof. Let us take a formula β with three levels of alternation and which starts with an existential block.

$$\beta = \exists x_1, ..., x_k \forall y_1, ..., y_l \exists z_1, ..., z_m B(x_1, ..., x_k, y_1, ..., y_l, z_1, ..., z_m)$$

We now give a satisifability-preserving $LTL[\mathsf{F}]{+}LEN^{un}$ formula $\widehat{\beta}$ such that β in Σ_3^P-SAT iff $\exists w, (w, 1) \vDash \widehat{\beta}$.

Take the first l prime numbers $p_1, ..., p_l$. Replace the y_js by $\ell \equiv 0 \mod p_j$. Let the resultant formula be called α. We give the formula $\widehat{\beta}$ below. It is a formula over the x and z propositions.

$$\hat{\beta} = \mathsf{G}(\ B[\ell \equiv 0 \mod p_j / y_j]) \wedge \mathsf{F}(\wedge_{j=1}^{l} \ell \equiv 0 \mod p_j) \wedge \bigwedge_{i=1}^{k} (\mathsf{G}x_i \vee \mathsf{G}\neg x_i)$$

Thanks to the prime number thorem we do not have to search too far (By the prime number theorem, asymptotically there are l primes within $l \log l$ and hence finding them can be done in polynomial time.) for the primes, and primality testing can be done in polynomial time.

Suppose the quantified boolean formula β is satisfiable. Then there is an assignment v to the x_is which makes the Π_2 subformula ($\forall\exists$ part) true. Consider the formula $\gamma = \beta[v(x_i)/x_i]$. We can represent an assignment to the y_js by an l length bit vector. There are 2^l different bit vectors possible. For each bit vector s we can obtain the formula γ_s, by substituting the y_js with the values from s. But since β is satisfiable, each of the γ_ss are satisfiable. Hence for all these formulae there is a satisfying assignment $Z^s : [m] \to \{0, 1\}$ to the variables z_r, for $r = 1, m$.

We are going to construct a word model M which will satisfy $\widehat{\beta}$. Take its length to be $n \geq \Pi_{j=1}^{l} p_j$ so that the future requirement is satisfied (2^{nd} formula). In every state of the word, let the proposition x_i take the value $v(x_i)$. Now we define at state t the valuation of $z_r, r = 1, m$, as follows. Let s be the bitstring represented by ($t \mod p_1 = 0, t \mod p_2 = 0, ..., t \mod p_l = 0$). Set the evaluation of z_r in the t^{th} state of M to be $Z^s(r)$.

Once we do this for all $t \leq n$, we find that $M, t \vDash \beta[\ell \equiv 0 \mod p_j/y_j][v(x_i)/x_i]$. And because $n \geq \Pi_{j=1}^{l} p_j$ we have $M, 1 \vDash \widehat{\beta}$. We have thus shown that there exists a word model satisfying $\widehat{\beta}$.

For the converse, suppose there is a word model M of length n which satisfies $\widehat{\beta}$. Then $n \geq \Pi_{j=1}^{l} p_j$. Set a valuation v for the x's as $v(x_i) = true$ iff $M, 1 \vDash x_i$. We have to now show that the formula $\gamma = \beta[v(x_i)/x_i]$ is satisfiable for all 2^l assignments to the y_js. That is, for all 2^l assignments to the y_j's there is an assignment to the z_rs which make γ true. Suppose s is a bitstring of length l representing an arbitrary assignment to the y_j's. Take a $t \leq n$, such that s equals the bitstring $(t \ mod \ p_1 = 0, t \ mod \ p_2 = 0, ..., t \ mod \ p_l = 0)$. Such a t exists because n is long enough. Let $Z^s(r)$ be the valuation of the z_r in the t^{th} state of M. This assignment to z_r makes the formula α true when the y_j's are assigned according to s. Hence β is satisfiable. □

5 Discussion

We observed in this paper that when LTL is extended with modulo counting, it does not matter if the specification of the moduli is in succinct notation. More generally this holds for computation within a finite symmetric group. This seems to have escaped the notice of verification researchers until now.

Are there other families of automata, where a "standard" enumeration of their states and transitions can be represented in logarithmic notation, and for which the PSPACE bound will continue to hold?

Motivated by the recent work of Laroussinie, Meyer and Petonnet, we also ask how far these ideas can be extended for pushdown systems.

References

1. Augustin Baziramwabo, Pierre McKenzie, and Denis Thérien. Modular temporal logic. In *Proc. 14th LICS*, page 344. IEEE, 1999.
2. E. Allen Emerson and E.M. Clarke Jr. Using branching time temporal logic to synthesize synchronization skeletons. *Sci. Comp. Program.*, 2:241–266, 1982.
3. Kousha Etessami, Moshe Y. Vardi, and Thomas Wilke. First-order logic with two variables and unary temporal logic. *Inf. Comput.*, 179(2):279–295, 2002.
4. Kit Fine. Propositional quantifiers in modal logic. *Theoria*, 36:336–346, 1970.
5. M.J. Fischer and R.E. Ladner. Propositional dynamic logic of regular programs. *J. Comp. Syst. Sci.*, 18(2):194–211, 1979.
6. David Harel, Dexter C. Kozen, and Rohit J. Parikh. Process logic: expressiveness, decidability, completeness. *J. Comp. Syst. Sci.*, 25:144–170, 1982.
7. David Harel and Rivi Sherman. Dynamic logic of flowcharts. *Inf. Contr.*, 64(1-3):119–135, 1985.
8. Jesper G. Henriksen and P.S. Thiagarajan. Dynamic linear time temporal logic. *Ann. Pure Appl. Logic*, 96(1-3):187–207, 1999.
9. I. N. Herstein. *Topics in Algebra*. Blaisdell, 1964.
10. François Laroussinie, Antoine Meyer, and Eudes Petonnet. Counting CTL. In *Proc. Fossacs*, LNCS, page to appear, 2010.

11. Hiroakira Ono and Akira Nakamura. On the size of refutation kripke models for some linear modal and tense logics. *Studia Logica: An International Journal for Symbolic Logic*, 39(4):325–333, 1980.
12. Olivier Serre. Vectorial languages and linear temporal logic. *Theoret. Comp. Sci.*, 310(1-3):79–116, 2004.
13. A.P. Sistla and E.M. Clarke Jr. The complexity of propositional linear temporal logics. *J. ACM*, 32(3):733–749, 1985.
14. Howard Straubing, Denis Thérien, and Wolfgang Thomas. Regular languages defined with generalized quantifiers. *Inf. Comput.*, 118(3):389–301, 1995.
15. Moshe Y. Vardi. From philosophical to industrial logics. In *Proc. 3rd Indian Conf. Log. Appl.,Chennai*, volume 5378 of *LNAI*, pages 89–115, 2009.
16. Moshe Y. Vardi and Pierre Wolper. Reasoning about infinite computations. *Inf. Comput.*, 115(1):1–37, 1994.
17. Pierre Wolper. Temporal logic can be more expressive. *Inf. Contr.*, 56(1-2):72–99, 1983.
18. Pierre Wolper, Moshe Y. Vardi, and A.P. Sistla. Reasoning about infinite computation paths. In *Proc. 24th Found. Comp. Sci.,Tucson*, pages 185–194. IEEE, 1983.