A note on a question of Peled and Wilke regarding Stutter-Invariant LTL

Kousha Etessami
Bell Labs, Murray Hill, NJ
email: `kousha@research.bell-labs.com`

Abstract

Peled and Wilke, [PW97], showed that the stutter-invariant languages expressible in linear propositional temporal logic (LTL) are precisely those expressible in LTL without the "next"-time operator. They did so by providing a translation, τ, which converts a stutter-invariant LTL formula φ to an equivalent formula $\tau(\varphi)$ not containing the "next" operator. Their $\tau(\varphi)$ is exponentially larger than φ, with a worst case upper bound of $|\tau(\varphi)| \leq 2^{O(n \times k)} \times |\varphi|$, where n is the number of distinct propositions in φ, and k is the "next"-depth of φ.

They asked whether this exponential blowup can be avoided. We give an improved translation, τ', which incurs a worst case blowup of $|\tau'(\varphi)| \leq n^{O(k)} \times |\varphi|$. This remains exponential in the "next"-depth of φ. However, if formulas are viewed as circuits, where identical subformulas are shared, and size is measured in terms of the number of distinct subformulas, one gets the following bound: $|\tau'(\varphi)| \leq O(|\varphi| + n^2)$.

The computation complexity of major algorithms based on LTL, e.g., model checking and translation to automata, is only affected by the number of distinct subformulas. So, without loss of efficiency, one can work with LTL circuits rather than formulas for those purposes. As a particular application, our upper bound on circuit size gives an alternative proof of the PSPACE upper bound due to [PWW98] for checking stutter-invariance of a temporal property: simply apply the well known (and by now heavily optimized) algorithms for testing validity of LTL formulas ([SC85, VW86]) to the circuit for $\varphi \leftrightarrow \tau'(\varphi)$.

We briefly recall the setting (see [PW97, Ete99]). Given atomic propositions $P = \{p_1, \ldots, p_n\}$, a Linear Temporal Logic (LTL) formula over P is built up from the atomic propositions using boolean connectives, $\neg, \vee$, and temporal operators X ("next") and U ("Until").[1]

An LTL formula, φ, defines a language $L(\varphi) \subseteq \Sigma^\omega$ of ω-words over the alphabet $\Sigma = 2^P$, according to the standard semantics for boolean connectives and temporal operators. We will say that two LTL formulas φ_1 and φ_2 are *equivalent* if $L(\varphi_1) = L(\varphi_2)$. As in [PW97], we let LTL(U) denote the fragment of LTL without the X operator.

[1] Other familiar temporal operators, like $\Box$ ("Always"), can be expressed using U and boolean connectives. Note that [PW97] uses the symbol"G" instead of $\Box$.

An ω-word $w = w_0w_1w_2 \ldots \in \Sigma^\omega$ is *stutter-free* if for all $i \in \mathbb{N}$, either $w_i \neq w_{i+1}$, or $w_i = w_j$ for all $j \geq i$. We call a mapping $f : \mathbb{N} \mapsto \mathbb{N}_{>0}$ from the natural numbers to the positive natural numbers a *stuttering function*. For a character $a \in \Sigma$, we write a^k to denote the concatenation $aa \ldots a$ of k copies of a. Given an ω-word $w = w_0w_1 \ldots \in \Sigma^\omega$, and a stuttering function f, let $w[f]$ denote the word $w_0^{f(0)}w_1^{f(1)} \ldots$. A language $L \subseteq \Sigma^\omega$ of ω-words is *stutter invariant*[2] if for all ω-words w and all stuttering functions f, $w \in L$ if and only if $w[f] \in L$.

Two stutter-invariant languages are identical if and only if they contain precisely the same stutter-free words, [PW97]. Lamport, [Lam83], originally observed that temporal logic without the X operator can only express stutter invariant properties. Hence if $L(\varphi)$ is stutter invariant, then for $\psi \in \mathrm{LTL}(U)$, ψ and φ are equivalent iff they agree on stutter-free words.

Given an LTL formula φ, [PW97] inductively obtain $\tau(\varphi) \in \mathrm{LTL}(U)$ such that $\tau(\varphi)$ agrees with φ on all stutter-free words. Let $V = \{v \mid v : P \mapsto \{true, false\}\}$ denote the set of valuations of all propositions. For $v \in V$, let B_v denote the unique conjunction of literals over P satisfied by only the valuation v. Their translation $\tau(\varphi)$ is given inductively by:

$$\tau(\varphi) = \begin{cases} \varphi & \text{if } \varphi \in P \\ \neg\tau(\psi) & \text{if } \varphi = \neg\psi \\ \tau(\psi)\ U\ \tau(\gamma) & \text{if } \varphi = \psi\ U\ \gamma \\ \bigvee_{v \in V}(\Box B_v \wedge \tau(\psi)) \vee \bigvee_{v \neq v' \in V}(B_v U (B_{v'} \wedge \tau(\psi))) & \text{if } \varphi = X\psi \end{cases}$$

The correctness of their translation follows from the fact that over stutter-free words $X\varphi$ is equivalent to the statement: "the first time after now that some new event happens, φ must hold, or else (if nothing new ever happens) φ must hold right now". By "new event" here, we mean a character of $\Sigma = 2^P$ distinct from the currect character. They encode this statement by considering all possible events, and thus all valuations $v \in V$. Let $|\varphi|$ denote the number of symbols in φ. It is not hard to derive the bounds given in the abstract for $|\tau(\varphi)|$. Observe that the only case which causes a blowup is a subformula of the form $X\psi$. In particular, because there are 2^n valuations $v \in V$, their translation *always* incurs an exponential blowup in n for such a subformula.

The simple observation that improves their translation is that the statement in quotes above is equivalent to: "Either for some i a change in the value of p_i occurs in the first new event after now, at which point φ holds, or the value of each p_i never changes, and right now φ holds".

We encode this latter statement with the following translation, τ', whose inductive definition is identical to τ, except when φ has the form $X\psi$. Let $[n] = \{1, \ldots, n\}$. $\tau'(X\psi)$ is given by:

$$\begin{aligned} \tau'(X\psi) \quad = \quad & \bigvee_{i \in [n]} (p_i \wedge (p_i\ U\ (\neg p_i \wedge \tau'(\psi))) \wedge \bigwedge_{\{j \in [n] \mid j \neq i\}} (p_j\ U\ \neg p_i) \vee (\neg p_j\ U\ \neg p_i)) \\ & \vee (\neg p_i \wedge (\neg p_i\ U\ (p_i \wedge \tau'(\psi))) \wedge \bigwedge_{\{j \in [n] \mid j \neq i\}} (p_j\ U\ p_i) \vee (\neg p_j\ U\ p_i)) \end{aligned}$$

[2] This definition, given in [Ete99], differs from the one given in [PW97], but can easily be shown to be equivalent.

$$\vee \; (\bigwedge_{i\in[n]} (\Box\, p_i \vee \Box\, \neg p_i) \wedge \tau'(\psi))$$

Let k_φ be the *"next"-depth* of the formula φ. This is defined inductively, with $k_{X\psi} = k_\psi + 1$, as the maximal nesting depth of subformulas of the form $X\psi$ within φ. The size $|\tau'(X\psi)|$ is bounded by: $|\tau'(X\psi)| \leq (2n+1) \times |\tau'(\psi)| + O(n^2)$. From this one obtains:

Theorem 1 *The translation τ' converts stutter invariant LTL formulas to equivalent* LTL(U) *formulas, such that for all φ: $|\tau'(\varphi)| \leq n^{O(k_\varphi)} \times |\varphi|$.*

Note the only use of τ' itself in the inductive definition of $\tau'(X\psi)$ is many repeated instances of $\tau'(\psi)$. This leads to the following notion. We can view the formulas $\tau'(\varphi)$ and φ more compactly as *circuits*, where identical subformulas are not repeated. An LTL circuit is a directed acyclic graph, with the following additional requirements: the edges out of each node are totally ordered, there is a unique source node, sink nodes are labeled with propositions, and all other internal nodes are labeled with boolean connectives or temporal operators. Each internal node corresponds to the subformula defined by applying its label to the subformulas defined by its (ordered) children. The root defines the entire formula. It easy to transform any LTL formula to an equivalent LTL circuit, where identical subformulas are identified.

Viewing LTL this way does not affect the computational complexity of the major algorithms based on it. In particular, the complexity of model checking, given an LTL circuit φ, remains PSPACE-complete in the size of φ (i.e., the number of nodes in the DAG representing φ).

Our translation remains exactly the same over circuits. We overload τ' to also denote our translation function for circuits. Since the subformulas defining "the next new event" need only be written once, and require only $O(n^2)$ size, we can derive:

Theorem 2 *The translation τ' converts stutter invariant LTL circuits to equivalent* LTL(U) *circuits, such that for all circuits φ: $|\tau'(\varphi)| \leq O(|\varphi| \times n^2)$.*

Note that the formula (circuit) $\tau'(\varphi)$ can always be computed efficiently, i.e., in time polynomial in the size of the input and output combined.

As commented in the abstract, one direct consequence of our bound is a PSPACE upper bound for determining whether an LTL formula defines a stutter-invariant property: one simply has to check the validity of the *circuit* form of $\varphi \leftrightarrow \tau'(\varphi)$, long known to be doable in PSPACE ([SC85]). The former result was obtained as a by-product of a more general result in [PWW98], relying on the fact that the congruence defined by stutter-equivalence is generated by a sequential relation. Our bound gives an alternative proof of this result, and allows one to avoid using a specialized algorithm for checking stutter-invariance, applying instead the well-optimized algorithms already available for testing validity of LTL formulas (or circuits).

Acknowledgment. Thanks to Doron Peled and to the anonymous referees for helpful comments.

References

[Ete99] K. Etessami. Stutter-invariant languages, ω-automata, and temporal logic. In *Proceedings of the 11th International Conference on Computer Aided Verification*, pages 236–248, 1999.

[Lam83] L. Lamport. What good is temporal logic. In R. E. A. Mason, editor, *Information Processing '83: Proc. IFIP 9th World Computer Congress*, pages 657–668, 1983.

[PW97] D. Peled and Th. Wilke. Stutter-invariant temporal properties are expressible without the next-time operator. *Information Processing Letters*, 63:243–246, 1997.

[PWW98] Doron Peled, Thomas Wilke, and Pierre Wolper. An algorithmic approach for checking closure properties of temporal logic specifications and ω-regular languages. *Theoretical Computer Science*, 195(2):183–203, 30 March 1998.

[SC85] A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM*, 32(3):733–749, 1985.

[VW86] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In Dexter Kozen, editor, *First Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 322–331, 1986.