Stutter-Invariant Languages, ω-Automata, and Temporal Logic

Kousha Etessami

Bell Labs
Murray Hill, NJ 07974
kousha@research.bell-labs.com

Abstract. Temporal logic and ω-automata are two of the common frameworks for specifying properties of reactive systems in modern verification tools. In this paper we unify these two frameworks in the linear time setting for the specification of stutter-invariant properties, which are used in the context of partial-order verification.
We will observe a simple variant of linear time propositional temporal logic (LTL) for expressing exactly the stutter-invariant ω-regular languages. The complexity of, and algorithms for, all the relevant decision procedures for this logic remain essentially the same as with ordinary LTL. In particular, satisfiability remains PSPACE-complete and temporal formulas can be converted to at most exponential sized ω-automata. More importantly, we show that the improved practical algorithms for conversion of LTL formulas to automata, used in model-checking tools such as SPIN, which typically produce much smaller than worst-case output, can be modified to incorporate this extension to LTL with the same benefits. In this way, the specification mechanism in temporal logic-based tools that employ partial-order reduction can be extended to incorporate all stutter-invariant ω-regular properties.

1 Introduction

Today, ω-automata and ω-regular languages are used in verification tools to both describe the models of reactive systems as well as to specify the properties being verified. While ω-automata typically form the basis for describing the system, a popular alternative for specifying properties is temporal logic [16], which offers an intuitive language for describing relationships between the occurrence of events over time. Linear time temporal logic, which we deal with in this paper, is used as a specification language in tools including SPIN [6], as well as more recent versions of SMV [12]. Standard linear time propositional temporal logic (LTL) can only express a strict subset of the ω-regular properties. To correct this, various remedies have been proposed (see [23]).

Stutter-invariant languages ([9]) are used in the context of partial-order verification [21,5,7,14]. In order to take full advantage of partial-order reduction, the properties that are specified need to be *stutter-invariant* (formal definitions will be supplied later). One way to assure that only stutter-invariant properties are specified is to restrict linear temporal logic, LTL, to only those formulas without the “next” operator. This is precisely the approach currently employed in the tool SPIN [6], which exploits partial-order reduction, and uses “next”-free LTL to specify properties.

LTL without “next” has been shown to accept all stutter-invariant LTL expressible properties [15]. However, the fact remains that LTL can not express all ω-regular properties, and similarly, LTL without “next” can not specify all stutter-invariant ω-regular properties.

In this paper we provide a simple variant of LTL for expressing all the stutter-invariant ω-regular languages, based on an equally simple temporal logic for all ω-regular languages. Significantly, the complexity of the relevant decision procedures remain the same as with ordinary LTL. In particular, satisfiability remains PSPACE-complete, as does model checking for negated formulas. Equally important, a practical algorithm for converting LTL formulas to automata ([4]), used in the tool SPIN [6], which typically produces much smaller than worst-case output, can be converted to incorporate this extension to LTL with basically the same benefits. As described in the conclusion, a preliminary version of this translation has been implemented with satisfactory results.

The logics we will describe are variants of Existential Quantified Linear Propositional Temporal Logic (EQLTL). Wolper [23] considered extensions of LTL with operators based on automata and right-linear grammars. He showed that this extended logic defines exactly the ω-regular languages and has the desired complexity. However, the syntax of a logic augmented with grammars is cumbersome and more akin to automata specifications. Sistla, Vardi, and Wolper, [19], considered variants of Wopler’s logic as well as Quantified Propositional Linear Temporal Logic (QLTL), where they provided complexity bounds for the satisfiability problem for formulas in the quantifier alternation hierarchy, including EQLTL. They showed, as part of this hierarchy, that QLTL satisfiability has non-elementary complexity and EQLTL satisfiability is PSPACE-complete. Kupferman, [8], studied branching time temporal logics with existentially quantified propositions.

It follows from Büchi’s original theorem ([1]) that EQLTL already captures the ω-regular languages, and hence also all of QLTL. The properties of EQLTL as a logic, in particular the complexity of decision procedures for the logic as well as its simple syntactic normal forms, make it worthy of closer examination.

The main subject of this paper is a *stutter-invariant* version of EQLTL, which we call SI-EQLTL. We will show that SI-EQLTL expresses precisely the stutter-invariant ω-regular languages. In proving this, we will also provide a simple normal form for ω-automata that accept stutter-invariant languages. We will then describe an efficient translation algorithm from the logic into automata.

Formal definitions of all the mentioned notions are provided in the next section. Section 3 overviews EQLTL and its correspondence to the ω-regulars, preparing the way for section 4, where SI-EQLTL and stutter-invariant languages are considered. Section 5 covers the improved translation algorithm to automata, and in section 6 we conclude and describe an implementation of this translation.

Note: After publication of this paper as a technical report ([3]), I have been informed by D. Peled that in [17] A. Rabinovich has independently obtained a characterization of the stutter-invariant ω-regular languages in terms of Lamport's Temporal Logic of Actions ([10]). Although TLA semantics generally differ substantially from LTL, his results appear to amount to the fact that SI-QLTL captures the stutter-invariant languages. But the complexity of the critical decision procedures for SI-QLTL remain non-elementary because of nested negation, and entire the reason we focus on SI-EQLTL is because of the complexity of these procedures.

2 Definitions and Background

Let $\mathcal{L}^P_{LTL}$ denote the set of LTL formulas over propositional symbols $P = \{p_1, \ldots, p_r\}$. These are defined according to the following inductive rules:

– $p_i \in \mathcal{L}^P_{LTL}$ for each $p_i \in P$.
– $\neg\varphi$, $\varphi \vee \psi$, $\Diamond\varphi$, $\bigcirc\varphi$, $\varphi\mathcal{U}\psi \in \mathcal{L}^P_{LTL}$, for $\varphi, \psi \in \mathcal{L}^P_{LTL}$.

We extend LTL by allowing quantification over new propositions $q_1, q_2, \ldots$. We define both the existential and universal fragment of *Quantified (propositional) Linear Temporal Logic (QLTL).* Formulas in $\mathcal{L}^P_{EQLTL}$, and $\mathcal{L}^P_{AQLTL}$, are defined, respectively, by the following additional rule:

– $\exists q_1 \ldots \exists q_k \varphi \in \mathcal{L}^P_{EQLTL}$, for $\varphi \in \mathcal{L}^{P\cup Q}_{LTL}$, $k \in \mathbb{N}$.
– $\forall q_1 \ldots \forall q_k \varphi \in \mathcal{L}^P_{AQLTL}$, for $\varphi \in \mathcal{L}^{P\cup Q}_{LTL}$, $k \in \mathbb{N}$.

We define the semantics of EQLTL and AQLTL, over ω-words and over Kripke structures. Let our alphabet be $\Sigma_P = 2^P$. An **ω-word** $w = w_0 w_1 w_2 \ldots \in \Sigma_P^\omega$ is a sequence of characters $w_i \in \Sigma_P$, where i ranges over $\mathbb{N} = \{0, 1, 2, \ldots\}$. Since we allow quantification over new propositional variables, we will also allow enlargement of our alphabet. Given P', such that $P \subseteq P'$, and given a character $a \in \Sigma_{P'}$, we define $a|_P \doteq \{p_i \in P \mid p_i \in a\}$. Let w and w' be ω-words over the alphabets Σ_P and $\Sigma_{P'}$, respectively. We say that w' is an *extension* of w iff $w_i = w'_i|_P$ for all $i \in \mathbb{N}$.

Given a word $w = w_0 w_1 w_2 \ldots$, and given a position $i \in \mathbb{N}$, we let $(w, i) \models \varphi$ denote the fact that φ is true on w at position i, defined inductively as usual, with the following semantics for propositional quantification:

1. $(w, i) \models \exists q \varphi$ if there is an extension $w' \in (\Sigma_{P\cup\{q\}})^\omega$ of w such that $(w', i) \models \varphi$.
2. $(w, i) \models \forall q \varphi$ if for all extensions $w' \in (\Sigma_{P\cup\{q\}})^\omega$ of w, $(w', i) \models \varphi$.

A *language* over Σ is a set $L \subseteq \Sigma^\omega$. A formula φ is said to *express* the language $L(\varphi) = \{w \mid (w, 0) \models \varphi\}$. We let **EQLTL** stand for the languages $\bigcup_P \{L(\psi) \mid \psi \in \mathcal{L}^P_{EQLTL}\}$, and we assume analogous definitions for the other logics.

We will define a variant of EQLTL that captures precisely the *stutter-invariant* ω-regular languages. Before we define what stutter-invariance means, let us define the logic. A word w' is a **harmonious** extension of the word w, if w'

is an extension of w such that, for all $i \in \mathbb{N}$, if $w_i = w_{i+1}$ then $w'_i = w'_{i+1}$. We define a restricted quantifier $\exists^h q\varphi$, which differs from ordinary quantification in the following way:

1. $(w, i) \models \exists^h q\varphi$ iff there is a harmonious extension $w' \in (\Sigma_{P\cup\{q\}})^\omega$ of w such that $(w', i) \models \varphi$.
2. $(w, i) \models \forall^h q\varphi$ if for all harmonious extensions $w' \in (\Sigma_{P\cup\{q\}})^\omega$ of w, $(w', i) \models \varphi$.

We define *stutter-invariant* EQLTL (AQLTL), denoted **SI-EQLTL** (**SI-AQLTL**), by replacing existential (universal) quantification of propositions with the harmonious quantification defined by $\exists^h$ ($\forall^h$), and by disallowing the use of the $\bigcirc$ operator in formulas. Thus, SI-EQLTL formulas are those of the form: $\exists^h q_1, \ldots, q_k\varphi$ where φ is a $\bigcirc$-free LTL formula.

A *Kripke structure* $\mathcal{K} = (S, R, \kappa)$ over the alphabet 2^P is a set S of states, together with a transition relation $R \subseteq S\times S$, and a labeling function $\kappa : S \rightarrow 2^P$. Let $\pi = s_0s_1, \ldots \in S^\omega$ be a sequence of states of $\mathcal{K}$. Extending the definition of κ to sequences, the sequence π defines an ω-word $\kappa(\pi) \doteq \kappa(s_0)\kappa(s_1)\ldots \in \Sigma_P^\omega$. Given an initial state s_{init}, we will say that a sequence π is a *proper sequence* with respect to $(\mathcal{K}, s_{init})$ if $s_0 = s_{init}$ and $(s_i, s_{i+1}) \in R$, for all i. We now define what it means for a formula φ to be satisfied by a Kripke structure, given an initial state s_{init}:

– $(\mathcal{K}, s_{init}) \models \varphi$ if for every proper sequence π of $(\mathcal{K}, s_{init})$, $(\kappa(\pi), 0) \models \varphi$.

We now briefly recall the terminology for ω-regular languages and ω-automata. A *Büchi automaton* is $A = (Q, \Sigma, \delta, F, Q_{start})$, where Q is a set of states, Σ an alphabet, $\delta \subseteq Q \times \Sigma \times Q$ is a transition relation, $F \subseteq Q$ a set of final states, and $Q_{start} \subseteq Q$ is a set of start states. Given a word w, a *run* r of A on w is a sequence of states $q_0, q_1, \ldots$, such that $q_0 \in Q_{start}$, $q_i \in Q$, and $(q_i, w_i, q_{i+1}) \in \delta$, for all i. Let *inf*(r) denote the set of states that occur infinitely often in r. A is said to *accept* an ω-word w from the alphabet Σ if there is a run r of A on w, such that there is some state $q \in F$ which occurs infinitely often on this run, i.e. $F \cap$ *inf*$(r) \neq \emptyset$. This is called the *Büchi acceptance condition.* Let $\mathbf{L(A)}$ be the the set of ω-words accepted by A. The **ω-regular** languages are the class of languages $\{L(A)|$ A a Büchi automaton $\}$.

Another acceptance criterion defines a *Muller automaton* $A = (Q, \Sigma, \delta, \mathcal{F}, Q_{start})$. Here, instead of one set F of final states, we are given a collection $\mathcal{F} \subseteq 2^Q$, and the Muller acceptance condition states that there exists a run r of A on w such that *inf*$(r) \in \mathcal{F}$. It is easy to convert Büchi acceptance to Muller acceptance, and, conversely, it is a well known theorem of McNaughton [13] that Muller automata accept precisely the ω-regular languages.

Given a word $w = w_0w_1\ldots$, and given function $f : \mathbb{N} \mapsto \mathbb{N}^+$ from the natural numbers to the positive natural numbers, let:

$$w[f] \doteq w_0^{f(0)}w_1^{f(1)}w_2^{f(2)}\ldots$$

Here, w_i^n is shorthand for the concatenation of n copies of the character w_i. A language L is **stutter-invariant**[1] if, for any ω-words $w = w_0 w_1 \ldots w_i \ldots$, and for any f

$$w \in L \iff w[f] \in L$$

An ω-word w is *stutter-free* if for all $i \geq 0$, $w_i \neq w_{i+1}$ or $w_i = w_j$ for all $j \geq i$.

Proposition 1. *Let L and L' be stutter-invariant languages. Then $L' = L$ iff they contain exactly the same stutter-free words.*

The stutter-invariant ω-regular languages are a strictly larger class than the stutter-invariant LTL definable languages:

Proposition 2. *The stutter-invariant language, over $\Sigma = \{a, b\}$:*

$$L = \{w \mid \textit{the substring ab occurs an odd number of times in } w\}$$

is not LTL definable, but is ω-regular.

3 EQLTL and the ω-Regular Languages

It follows from the proof of Büchi's theorem and known results about LTL, that EQLTL captures exactly the ω-regular languages. In this section we will look carefully at this correspondence. This will facilitate our proof in the next section that a variant of this logic captures the stutter-invariant ω-regular languages.

Theorem 1. *(follows [1] & [18]. See also [19].) EQLTL defines exactly the ω-regular languages.*

Proof. $\Leftarrow$. Given an automaton $A = (Q = \{q_1, \ldots, q_k\}, \Sigma_P, \delta, F, s)$, we write an *EQLTL* formula that expresses $L(A)$. This is just the easy direction of Büchi's theorem: we "guess" an accepting run, and verify it, using temporal logic instead of first-order logic. We modify things slightly to facilitate the stutter-invariant version of this translation.

$$\varphi \equiv \exists q_1, \ldots, q_k (\Box(\bigwedge_{i \neq j} \neg(q_i \wedge q_j)) \wedge \tag{1}$$

$$(\bigvee_{\{(q_i,a,q_j) \in \delta \ \mid \ q_i \in Q_{start}\}} (\bigwedge_{p_l \in a} p_l \wedge \bigwedge_{p_r \notin a} \neg p_r) \wedge \ q_j) \ \wedge \tag{2}$$

$$\Box(\bigvee_{(q_i,a,q_j) \in \delta} (q_i \wedge \bigcirc(\bigwedge_{p_l \in a} p_l \wedge \bigwedge_{p_r \notin a} \neg p_r) \wedge \bigcirc q_j) \wedge \tag{3}$$

$$\bigvee_{q_j \in F} \Box \Diamond q_j \tag{4}$$

[1] This definition differs slightly from previous definitions in the literature, e.g., in [15], but is equivalent, and perhaps somewhat simpler conceptually because we avoid introducing the notion of stutter equivalence in order to get at the notion of stutter invariance.

$\Rightarrow$. Given an EQLTL formula $\varphi = \exists q_1, \ldots, q_k \psi$, let A_ψ be a Büchi automaton for ψ (see [18,22,2]). We construct A_φ, with the same set of states, by projection from A_ψ: for any edge $(Q, a, Q') \in \delta_{A_\psi}$, where $a = \{p_{i_1}, \ldots, p_{i_r}, q_{i_1}, \ldots, q_{i_l}\}$, put the edge (Q, a', Q') in δ_{A_φ}, where $a' = \{p_{i_1}, \ldots, p_{i_r}\}$. $\square$

Corollary 1. *(see [20]) Every EQLTL formula φ is equivalent to one in the following normal form (note only one quantified proposition):*

$$\varphi' = \exists q \ \psi$$

where ψ is an LTL formula without the "Until" operator $\mathcal{U}$.

Moreover, there is a polynomial p, such that for an automaton A, there is a normal form formula φ_A, $L(\varphi_A) = L(A)$, such that $|\varphi_A| = p(|A|)$.

The proof is an adaptation of [20]. We can readily eliminate the $\mathcal{U}$ operator because the formula φ in the proof of Theorem 1 contains none. Next we observe the computation complexity of the associated decision procedures.

Corollary 2.

1. *([19]) The satisfiability problem for EQLTL is PSPACE-complete.*
2. *An EQLTL formula φ can be translated to an equivalent Büchi automaton of size $2^{O(|\varphi|)}$, in time proportional to the size of the output.*
3. *Model checking, given a Kripke structure K and* the negation *of an EQLTL formula, or given an AQLTL formula, is PSPACE-complete.*

4 Stutter-Invariant ω-Regular Languages and SI-EQLTL

Now we prove a result analogous to Theorem 1 for SI-EQLTL. First, we provide a normal form for automata that accept a stutter-invariant language.

Definition 1. *A Muller automaton $A' = (Q', \delta', \{q'_{start}\}, \mathcal{F}')$ is a* **stutter-invariant (SI) automaton** *if every state is reachable from q'_{start}, and all of the following syntactic properties are satisfied, for each state q of A', $q \neq q'_{start}$:*

1. *All incoming edges to q are labeled with the same character, a_q.*
2. *$(q, a_q, q) \in \delta'$, and $(q, b, q) \notin \delta'$ for $b \neq a_q$.*
3. *$(q, a_q, q') \notin \delta'$ for $q' \neq q$.*
4. *Moreover, the exceptional start state q'_{start} has no incoming edges.*

A cleaner, equivalent, way to view such automata is as Kripke structures with (Muller) acceptance conditions, and a given set of start states. Viewed this way, an SI automaton then amounts to a Kripke structure with the following two additional properties:

1. Every state s has a self loop, i.e., $\forall\ s \in S$, $R(s, s)$.
2. $\forall\ s \neq s' \in S$, if $R(s, s')$ then $\kappa(s) \neq \kappa(s')$.

Proposition 3. *An SI automaton accepts a stutter-invariant language.*

Lemma 1. *If $L(A)$ is stutter-invariant then $L(A) = L(A')$, where A' is a SI automaton. Moreover, we can pick A' such that $|A'| \leq O(|A| \times |\Sigma|)$.*

Proof. We define A' to mimic A, except that a state of A' remembers the last-seen character. In addition, we need some extra surgery in order not to allow both arrival and departure from a state using the same character.

Given $A = (Q, \delta, Q_{start}, F)$, we define $A' = ((Q \times \Sigma) \cup \{q'_{start}\} \cup \{q_a^{new} \mid a \in \Sigma\}, \delta', \{q'_{start}\}, \mathcal{F}')$:

1. $((q,a), b, (q',c)) \in \delta'$ iff ($b = c \neq a$ and $(q,b,q') \in \delta$) or ($b = c = a$ and $q = q'$)
2. $((q,a), b, q_c^{new}) \in \delta'$ iff ($b = c \neq a$ and starting at state q in A there exists an accepting run on the word b^ω).
3. $(q_a^{new}, b, q_c^{new}) \in \delta'$ iff $a = b = c$.
4. $(q'_{start}, a, (q_2, b)) \in \delta'$ iff $a = b$ and $(q_1, a, q_2) \in \delta$ for some $q_1 \in Q_{start}$.

Now, the accepting sets are given by: $\mathcal{F}' = \{F' \subseteq Q \times \Sigma \| F'| > 1 \ \wedge \exists (q,a) \in F' \ s.t. \ q \in F\} \cup \{\{(q,a)\} \mid q \in F \ \wedge (q,a,q) \in \delta\} \cup \{\{q_a^{new}\} \mid a \in \Sigma\}$.

By inspection, A' is an SI automaton. By Proposition 3 and Proposition 1 we need only show that $L(A)$ and $L(A')$ contain the same stutter-free words.

Claim. For any stutter-free word w, there is an accepting run r of A on w if and only if there is an accepting run r' of A' on w.

We must omit the proof of the claim, which splits things into two cases: either (1) $w = w_0 w_1, \ldots w_i^\omega$, or (2) $w = w_0 w_1, \ldots$, where there are never two consecutive occurrences of the same character. The claim concludes the lemma, as A' satisfies all the required conditions. □

Theorem 2. *SI-EQLTL defines exactly the stutter-invariant ω-regular languages.*

Proof. $\subseteq$: First, a SI-EQLTL formula $\psi = \exists^h q_1, \ldots, q_k \varphi$ can only express a stutter-invariant language. To see this, consider w and $w[f]$ for any $f : \mathbb{N} \mapsto \mathbb{N}^+$.

Claim. $w \in L(\psi)$ if and only if $w[f] \in L(\psi)$.

Proof. $\Rightarrow$. Suppose $w \in L(\psi)$, then there is a harmonious extension v of w such that $v \in L(\varphi)$. But the $\bigcirc$-free LTL formula, φ, accepts a stutter-invariant language ([15]), and thus since $v[f]$ is a harmonious extension of $w[f]$, and $v[f] \in L(\varphi)$, it follows that $w[f] \in L(\psi)$.

$\Leftarrow$. Suppose $w[f] \in L(\psi)$. Thus there is a harmonious extension $v[f]$ of $w[f]$ such that $v[f] \in L(\varphi)$. But, since $L(\varphi)$ is stutter-invariant, it must again be the case that v is a harmonious extension of w such that $v \in L(\varphi)$, and hence $w \in L(\psi)$. □

Now, to see that $L(\psi)$ is ω-regular: Let $A_{guess} = (Q_{guess} = 2^{P \cup \{q_1, \ldots, q_k\}} \cup \{q_{start}\}, \delta_{guess}, \{q_{start}\}, Q_{guess})$ be an automaton which has a transition $\delta_{guess}(q, a, q')$ if and only if

1. $q' = a$, (note: states, as well as transition labels, are denoted by sets of propositions).
2. If $a|_P = q|_P$ then $a = q$. (this insures the harmonious nature of the guess).

Let the automaton A_φ for φ be derived from the tableau construction ([18, 22]). We construct the automaton $A_\psi \subseteq A_\varphi \times A_{guess}$, where the states are $Q_\psi = \{(g, f) \mid g \in Q_\varphi \ \wedge \ f \in Q_{guess}\}$, where moreover g and f are *consistent*, meaning that, for $q_i \in cl(\varphi)$, $q_i \in g \iff q_i \in f$. The transition relation $\delta((g, f), a, (g', f'))$ holds iff $\delta_\varphi(g, a, g')$ and $\delta_{guess}(f, a, f')$, the start states $Q_{\psi start} = \{(g, f) \mid \varphi \in g\}$, and $F_\psi = F_\psi \times F_{guess}$. It can be verified that the automaton A_ψ accepts $L(\psi)$, with A_{guess} basically used to insure that we only "guess" harmonious extensions.

$\supseteq$. Given $L(A)$, a stutter-invariant language, our objective is to write an SI-EQLTL formula expressing the language $L(A)$. By Lemma 1, we can assume that A is a stutter-invariant automaton. We will use the fact ([15]) that $\bigcirc$-free LTL captures precisely the stutter-invariant subset of LTL.

Consider an EQLTL formula $\psi = \exists q_1, \ldots, q_k \phi$ expressing the language L(A). The crucial point is that because A has the syntactic normal form, there exists an accepting run r of A on w iff there exists a harmonious extension w' of w, such that w' is satisfied by the LTL formula. Thus, it suffices to convert the quantification to $\exists^h q_1, \ldots, q_k$.

It only remains to remove the $\bigcirc$ operators from the expression. This can be done by extending the proof of [15]. Let **SI-EQLTL**($\bigcirc$) be the logic where we *do* allow the ($\bigcirc$) operator to occur in the LTL part of the formula.

Claim. Any SI-EQLTL($\bigcirc$) formula, ψ, that accepts a stutter-invariant language can be converted to an SI-EQLTL formula ψ' such that $L(\psi) = L(\psi')$.

The proof is as [15]. The only new observation needed is that any harmonious extension of a stutter-free word is also stutter-free. That concludes the theorem. □

To prove a normal form result for SI-EQLTL analogous to EQLTL, which eliminates the use of the binary $\mathcal{U}$ operator, we will need the following stutter-invariant version of the $\bigcirc$ operator, $\bigcirc^\star$, which intuitively means "at the next distinct character". Formally, let $\bigcirc^\star \phi$ be defined as follows:

– $(w, i) \models \bigcirc^\star \phi$ if $(\exists j > i w_j \neq w_i \wedge \forall\ i',\ i \leq i' < j w_i = w_{i'}) \rightarrow (w, j) \models \phi$.

Let SI-EQLTL($\bigcirc^\star$, $\not{\mathcal{U}}$) denote the variant of SI-EQLTL where the $\mathcal{U}$ operator is disallowed, but $\bigcirc^\star$ is allowed. It can be shown that

Corollary 3. *SI-EQLTL expresses the same languages as SI-EQLTL($\bigcirc^\star$, $\not{\mathcal{U}}$).*

We are unable to provide a normal form where only one existential quantification is necessary, because Thomas's elimination argument [20] doesn't work in the stutter-invariant setting. It will be interesting to establish whether such a normal form exists. Finally, we address the costs and complexities involved in the mentioned translations and results above. They are, as one would expect, basically the same as for EQLTL.

Corollary 4.

1. *The satisfiability problem for SI-EQLTL is PSPACE-complete.*
2. *An SI-EQLTL formula φ can be translated to an equivalent Büchi automaton of size $2^{O(|\varphi|)}$, in time proportional to the size of the output.*
3. *Model-checking, given a Kripke structure K and* the negation *of an SI-EQLTL formula, or given an SI-AQLTL formula, is PSPACE-complete.*

5 Improved Algorithm for SI-EQLTL Conversion to Automata

In translating from LTL formulas to automata, a naive implementation of the tableau construction always incurs the worst-case exponential blow-up. In [4] an algorithm is provided which in practice behaves much better. A version of this algorithm for $\bigcirc$-free LTL has been implemented in the SPIN tool.

For our purposes, it is important to know that, rather than a standard Büchi automaton $A = (Q, \delta, \mathcal{F}, s)$ over $\Sigma_P = 2^P$, the automaton produced by the algorithm of [4] actually has the following **special form**: for every state $q \in Q$, there is a unique *term* (a conjunction of literals) from P, such that every "edge" from another state to q has the label σ_q; this term is a symbolic shorthand for all the characters consistent with it, meaning the actual edge (q', a, q) exists iff a is consistent with the term σ_q. In practice, this shorthand can be much more concise than the ordinary notation for automata. Later, we will need another important fact about the output of the [4] algorithm, namely, a monotonicity which it preserves.

We now describe how to modify the [4] algorithm to work for translating from both EQLTL and SI-EQLTL to automata. For EQLTL, modifying the algorithm is trivial. The only observation necessary is that existential quantification corresponds to projection, even on term-labeled edges:

Proposition 4. *Given an EQLTL formula $\varphi = \exists q_1 \dots q_k\ \psi$, and given a special form automaton A_ψ, such that $L(\psi) = L(A_\psi)$, the special form automaton A_φ derived from A_ψ by removing all literals over $\{q_1, \dots, q_k\}$ from the terms labeling the edges of A_ψ, defines precisely the language defined by φ, i.e., $L(A_\varphi) = L(\varphi)$.*

Note that the automaton generated for $\exists q_1 \dots q_k \psi$ is never bigger that the one generated for ψ. The case of SI-EQLTL is more interesting and complicated. In particular, it is not in general possible to obtain an automaton for $\exists^h t_1 \dots t_k \psi$ which is no bigger than the automaton for ψ produced by the [4] algorithm.

The following theorem gives an algorithm that incurs exponential blow-up in terms of k and quadratic blow-up in terms of the number of unquantified propositions in ψ. We then give a modification of this algorithm which in practice behaves much better.

Theorem 3. *Given a formula $\varphi = \exists^h t_1 \dots t_k \psi$, where ψ is a formula over the propositions $p_1 \dots, p_r, t_1, \dots, t_k$, there is a special form automaton A_φ, such that $L(A_\varphi) = L(\varphi)$, and such that*

$$|A_\varphi| \leq O(2^k \times r^2 \times |A_\psi|)$$

where A_ψ is a special form automaton for the formula ψ (as produced by the [4] algorithm).

Proof. Let $T = \{t_1, \ldots, t_k\}$. Given the set $P = \{p_1, \ldots, p_r\}$ of propositions, let $\neg P$ denote the set $\{\neg p_1, \ldots, \neg p_r\}$ of negations of these propositions. Given a special form automaton $A_\psi = (Q, \delta, \mathcal{F}, q_{start})$ for ψ, we would like to obtain an automaton for φ.

We now define an automaton A' which will be central to the definition of A_φ.

$$A' = (Q' \subseteq (Q \times 2^{\{t_1,\ldots,t_k\}} \times (P \cup \neg P)^2) \cup \{s_{start}\}, \delta', \mathcal{F}', s_{start})$$

The states Q' of A', other than the start state s_{start}, consist of those triplets (q, θ, τ), where $q \in Q$ is a state of A_ψ, θ is a subset of T and defines a *full* valuation of all the variables in T, and τ specifies two elements of the set $P \cup \neg P$, where these two elements are consistent with each other, i.e., it is not the case that one is the negation of the other. Furthermore, for (q, θ, τ) to qualify as a state in Q' it must also satisfy the following extra condition: we view τ as specifying a partial valuation of P, namely, the two literals specified in τ must hold. Now, the extra condition that must be satisfied by (q, θ, τ) is that the unique term σ_q, which labels all edges in A_ψ which enter the state q, must be consistent with the valuation (full on T and partial on P) specified by θ and τ. A' will also be a special form automaton, in that all edges into a state $s = (q, \theta, \tau)$ will be labeled with the same term σ_s.

The transition $\delta'(s' = (q', \theta', \tau'), \sigma_s, s = (q, \theta, \tau))$ will exist if and only if all of the following conditions hold:

1. σ_s is consistent with the valuation defined by θ, and the partial valuation on the pair of p_j's defined by τ.
2. There is a transition $\delta(q', \sigma_q, q)$ in A_ψ, such that the term σ_q is a *subterm* of the term σ_s, meaning that every literal of σ_q is also a literal of σ_s.
3. If the valuations θ' and θ on the t_i's are inconsistent, then the partial valuations defined by τ' and τ must also be inconsistent.

The transitions out of the exceptional start state s_{start} are defined as follows $\delta'(s_{start}, \sigma_s, (q, \theta, \tau))$ holds if and only if the first two condition above hold, with q_{start} substituted for q' in the statement of the second condition. Next we define the acceptance condition $\mathcal{F}'$ in A'. For each set $F = \{q_{i_1}, \ldots, q_{i_r}\} \in \mathcal{F}$, we put the set $F' = \{s = (q_{i_j}, \theta, \tau) \mid j \in \{1, \ldots, r\} \ \wedge \ s \in Q'\}$ in $\mathcal{F}'$.

From the automaton A' we will now obtain the automaton A_φ, by "projecting out" the q_i's as we normally would for regular existential quantification. More formally: for each transition $\delta'(s', \sigma_s, s)$ of A' the term σ_s is replaced in A_φ by a term σ_s^{new} with all the literals over the q_i's removed, obtaining the transition $\delta_\varphi(s', \sigma_s^{new}, s)$. It remains to show that A_φ defines the language we are after.

Lemma 2. *Given $\exists^h t_1, \ldots, t_r \psi$, and given A_φ obtained from φ via the above construction:*

$$L(A_\varphi) = L(\varphi)$$

We have to omit the proof, which is a bit technical, but the reason we need only remember two literals from $(P \cup \neg P)$ is that we can "guess" the literals over P which will distinguish distinct consecutive characters, and we need two literals rather than one per character because we use one to distinguish it from its predecessor and the other to distinguish it from its successor. This concludes our theorem. $\square$

The construction of A' was the crucial step in Theorem 3, and it is there that a $2^k \times r^2$ blow-up occurs. As it stands, A' *always* suffers from this blow-up. However, with an important observation, we can modify the algorithm so that the blow-up in k need not be worst-case.

In going from A_ψ to A', for every state q of A_ψ, and every τ, we built the states (q, θ, τ) for every *full* valuation θ of T. The observation is that these θ's need not be full valuations of T. In fact, the only condition we need is that, for any run of A_ψ, the domains of the sequence of expansions θ_i's in our new automaton form a *monotonic* (non-increasing) sequence of partial valuations of T, by which we mean that the domains form a monotonic sequence of sets. This assures us that, if there is any accepting run in the resulting automaton, then an accepting run can be constructed where the "guessed" variables form a harmonious extension. The reason, intuitively, is that we can safely extend a partial valuation of T that is consistent with a predecessor to fully match this predecessor valuation, without forcing an inconsistency with future valuations, because future valuations can evaluate at most the same or fewer variables and can thus be extended likewise if they are consistent with their predecessor.

But how are we to come up with such a monotonically decreasing sequence of partial-valuations to satisfy the condition? It turns out that we are in a rather fortuitous situation. The output of [4]'s algorithm already provides us with such a sequence. Each state of that output is marked by a set of subformulas of the original temporal logic formula being translated, and here is where we find our monotonic sequence: on any path in the automaton the set of t_i's that occur in the set of subformulas in each node form a monotonic (non-increasing) sequence. We can thus simply use these sets as our monotonic sequence directly from the [4] algorithm. Using this observation, we can reduce the state space of A' by only evaluating the $t_i's$ that need to be evaluated for each given state q of A_ψ. We have to omit details.

6 Conclusions

We have provided a simple temporal logic, SI-EQLTL, for expressing the stutter-invariant ω-regular languages. We have shown that the basic algorithms for, e.g., satisfiability and conversion to automata, for LTL, can be modified to the setting of SI-EQLTL without any substantial penalty in computational complexity. Along the way, we have defined stutter-invariant automata, a syntactic normal form for automata which captures the stutter-invariant ω-regular languages.

The purpose of such a logic is to close the gap, in a natural way, between systems like COSPAN where properties are specified as ω-automata, and those such as SPIN, where properties are specified in the weaker LTL formalism, and where, moreover, only stutter-invariant properties are allowed in order to enable partial-order verification.

One potential criticism for both SI-EQLTL and EQLTL is that, although both logics are semantically closed under complementation, they are not *syntactically* so. Indeed, complementation of a formula is, in the worst case, costly: incurring an exponential blow-up. As a result, in order to perform model checking with the same complexity as LTL, we need to work with the *negation* of SI-EQLTL formulas or with SI-AQLTL formulas. Although this is undesirable, it is a situation no different than the behavior of non-deterministic ω-automata, for which complementation is similarly costly. This was the reason behind [11]'s advocacy of $\forall$-automata. In other words, in either formalism one has to deal with the cost of complementation, but the benefits of a more succinct logical representation make these temporal logics an attractive alternative to automata.

We have implemented the translation algorithm of section 5 in the programming language ML, extending an implementation due to Doron Peled of the [4] algorithm. Preliminary experiments indicate that the translation produces very reasonable sized automata. The intention has been to ultimately incorporate the extended logic, using the translation, in Gerard Holzmann's tool SPIN in order to supply it with the extra expressive power.

Acknowledgment. Thanks to Mihalis Yannakakis for several important comments. Thanks to Doron Peled for allowing me to use his ML implementation of the [4] algorithm. Thanks to Gerard Holzmann, Orna Kupferman, Bob Kurshan, Doron Peled, Moshe Vardi, and Thomas Wilke for helpful comments.

References

1. J. R. Büchi. On a decision method in restricted second-order arithmetic. In *Proceedings of the International Congress on Logic, Methodology, and Philosophy of Science*, 1960. Stanford University Press, 1962.
2. E. A. Emerson. Temporal and modal logics. In J. van Leeuwen, editor, *Handbook of Theoret. Comput. Sci.*, volume B, pages 995–1072. Elsevier, Amsterdam, 1990.
3. K. Etessami. Stutter-invariant languages, ω-automata, and temporal logic. Technical Report BL011272-980611-07TM, Bell Laboratories, June 11 1998.
4. R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *PSTV95, Protocol Specification Testing and Verification*, pages 3–18, 1995.
5. P. Godefroid and P. Wolper. A partial approach to model checking. In *Proc. 6th Ann. IEEE Symp. on Logic in Computer Science*, pages 406–415, 1991.
6. G. J. Holzmann. The model checker spin. *IEEE Transactions on Software Engineering*, 23(5):279–295, 1997.
7. G. J. Holzmann and D. Peled. An improvement in formal verification. In *7th International Conference on Formal Description Techniques*, pages 177–194, 1994.
8. O. Kupferman. Augmenting branching temporal logics with existential quantification over atomic propositions. *Journal of Logic and Computation*, 7:1–14, 1997.

9. 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.
10. L. Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, pages 872–923, 1994.
11. Z. Manna and A. Pnueli. Specification and verification of concurrent programs by ∀-automata. In *Proc. 14th Ann. ACM Symp. on Principles of Programming Languages*, pages 1–12, 1987.
12. K. McMillan, 1998. See http://www-cad.eecs.berkeley.edu/kenmcmil for recent versions of SMV and its documentation.
13. R. McNaughton. Testing and generating infinite sequences by a finite automaton. *Information and Control*, 9:521–530, 1966.
14. D. Peled. Combining partial order reductions with on-the-fly model checking. *Formal Methods in System Design*, 8:39–64, 1996.
15. D. Peled and T. Wilke. Stutter-invariant temporal properties are expressible without the next-time operator. *Information Processing Letters*, 63:243–246, 1997.
16. A. Pnueli. The temporal logic of programs. In *Proc. 18th Symp. on Foundations of Computer Science*, pages 46–57, 1977.
17. A. Rabinovich. Expressive completeness of temporal logic of actions. In *Mathematical Foundations of Computer Science*, pages 229–238, August 1998.
18. A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM*, 32(3):733–749, 1985.
19. A. P. Sistla, M. Y. Vardi, and P. Wolper. The complementation problem for büchi automata with applications to temporal logic. *Theoretical Computer Science*, 49:217–237, 1987.
20. Thomas, W. Classifying regular events in symbolic logic. *Journal of Computer and System Sciences*, 25:360–376, 1982.
21. A. Valmari. A stubborn attack on state explosion. *Formal Methods in System Design*, 1:297–322, 1992.
22. M. Y. Vardi and P. Wolper. Reasoning about infinite computation. *Information and Computation*, 115:1–37, 1994.
23. Pierre Wolper. Temporal logic can be more expressive. *Information and Control*, 56:72–99, 1983.