Counting LTL

François Laroussinie[1] Antoine Meyer[2]
Eudes Petonnet[1]

[1] LIAFA, Université Paris Diderot – Paris 7 & CNRS UMR 7089, France
{Francois.Laroussinie,Eudes.Petonnet}@liafa.jussieu.fr
[2] LIGM, Université Paris Est – Marne-la-Vallée & CNRS UMR 8049, France
Antoine.Meyer@univ-mlv.fr

Abstract

This paper presents a quantitative extension for the linear-time temporal logic LTL allowing to specify the number of states satisfying certain sub-formulas along paths. We give decision procedures for the satisfiability and model checking of this new temporal logic and study the complexity of the corresponding problems. Furthermore we show that the problems become undecidable when more expressive constraints are considered.

1. Introduction

Temporal logic (TL) is a well-known and well-studied formalism for specifying and verifying properties of automated systems [13]. Classical temporal logics, such as LTL or CTL, express properties on the temporal ordering of events along the executions (see [6] for a survey). Many extensions of these formalisms have been studied, whose aim is usually to improve expressivity in order to capture more complex specifications, or to make properties shorter and easier to write. Of course there is an important trade-off between the expressivity of the logic and the efficiency of decision procedures, the ultimate goal being to algorithmically decide satisfiability or model-checking problems.

Among the well-known extensions of classical temporal logics, we can mention real-time TLs (see for example [2, 3, 7]) where it is possible to add timing constraints in properties (for example, one can specify that an event A follows an event B in at most 5 time units) or probabilistic TLs (see [10] for such an extension of CTL) where it is possible to express properties like "event A will occur with probability 0.99 or greater". These two extensions are called *quantitative extensions*. Another classical variant consists in adding some form of regular expressions [8] (or operators defined with grammars as in [17]).

In this paper, we present a counting extension of LTL in the line of [12], called CLTL, where *Until* modalities are equipped with constraints on the number of true occurrences of certain subformulas. For instance, in a mutual exclusion protocol where two processes try to access the same critical section, the formula $\mathsf{G}\big(\mathsf{req}_1 \Rightarrow \mathsf{F}_{[\sharp \mathsf{cs}_2 \leq 5]}\mathsf{cs}_1\big)$ expresses the fact that whenever process 1 requests access to the critical section it is eventually granted access, and until then process 2 can be granted access at most 5 times. More generally, we allow constraints to be arbitrary Boolean combinations of atomic statements of the form $\sum_i \alpha_i \cdot \sharp\varphi_i \sim c$, where c and each α_i are positive integers, $\sim$ is a comparison operator and each $\sharp\varphi_i$ represents the number of states from which some *arbitrary* CLTL formula φ_i holds along a certain prefix of the run.

We show that, even though CLTL formulas can be translated into classical LTL, this translation might yield an exponential blow-up in formula size. We then turn to the satisfiability and model-checking problems for CLTL, for which we provide automata-based algorithms running in exponential space. This complexity is asymptotically optimal, since both problems turn out to be EXPSPACE-complete. We conclude this algorithmic study by presenting a fragment of CLTL whose satisfiability and model-checking problems are PSPACE-complete, and show that any generalization of constraints with subtraction makes both problems undecidable. Finally, we show that for a similar counting extension of CTL*, the model-checking problem remains solvable in EXPSPACE.

This work is related to our previous effort on counting extensions of CTL [12], where we use the same counting constraints as described above. By varying the allowed syntax of constraints, we presented a thorough account of the expressiveness and succinctness of the logic with respect to CTL, and proposed an algorithmic study of the model-

checking problem, which ranges from P-complete when only atomic constraints are considered to Δ_2^{P}-complete for the full logic. Contrary to CLTL, we also managed to characterize decidable fragments with subtractive constraints.

There have also been several works on extensions of LTL to handle quantitative aspects of systems. In [8], the authors extend linear-time logic with some simple regular expressions along with quantitative constraints on the number of occurrences of sub-expressions. They present algorithms for model-checking this extension, whose time complexity is exponential in the size of formulas and the *value* of integer constants (satisfiability is not considered). This means their time complexity is doubly-exponential, which is comparable to ours. However, our logic can more easily express complicated quantitative constraints but cannot specify as simply properties on the order of events. Another interesting specification language is Sugar/PSL [14], which defines many additional operators above LTL. These include in particular some counting constraints which are used together with regular expressions, subsuming CLTL with atomic constraints. To our knowledge, there is no accurate study of lower complexity bounds for these extensions [4].

The paper is organized as follows. Section 2 defines the logic CLTL, whose expressivity and succinctness are studied in Section 3. Section 4 presents an EXPSPACE satisfiability algorithm based on alternating Büchi automata, as well as the EXPSPACE-hardness proof and a PSPACE algorithm for a fragment of CLTL. Section 5 deals with an undecidable extension of the constraint language. Finally Section 6 presents a counting extension of CTL^*.

2. Definitions

Models. Let AP be a set of atomic propositions. In linear-time temporal logics, formulas are interpreted over infinite words in $(2^{\mathsf{AP}})^\omega$. Given such a word w, w_i denotes the i-th letter and w^i is the i-th non-empty suffix of w with $i \geq 0$. As we will be considering the model-checking problem, we also recall the classical notion of Kripke structure:

Definition 1. A *Kripke structure* (or KS) $\mathcal{S}$ is a tuple $\langle Q, q_{\text{init}}, R, \ell \rangle$ where Q is a finite set of states, $q_{\text{init}} \in Q$ is the initial state, $R \subseteq Q \times Q$ is a total accessibility relation and $\ell : Q \to 2^{\mathsf{AP}}$ is a labeling of states with atomic propositions.

A run (or path) ρ of $\mathcal{S}$ is an infinite sequence of states $q_0 q_1 q_2 \ldots$ such that $(q_i, q_{i+1}) \in R$ for every i. We use $\rho(i)$ to denote the state q_i and ρ^i to denote the suffix $q_i \cdot q_{i+1} \cdots$ of ρ. $\mathsf{Runs}(q)$ denotes the set of runs starting from some state $q \in Q$ and $\mathsf{Runs}(\mathcal{S})$ is a shortcut for $\mathsf{Runs}(q_{\text{init}})$.

In the following, we will be referring to both infinite words in $(2^{\mathsf{AP}})^\omega$ and paths of some KS as *runs*.

Counting LTL. We define a quantitative extension of LTL able to express constraints over the number of times certain sub-formulas are satisfied along a run:

Definition 2. Given a set of atomic propositions AP, we define:

$$\mathsf{CLTL} \ni \varphi, \psi \ ::= \ P \mid \varphi \wedge \psi \mid \neg\varphi \mid \varphi \mathsf{U}_{[C]} \psi$$

where $P \in \mathsf{AP}$ and C is a counting constraint defined as:

$$\mathcal{C} \ni C, C' \ ::= \ \top \mid C \wedge C' \mid \neg C \mid \textstyle\sum_i \alpha_i \cdot \sharp\psi_i \sim k$$

where $k, \alpha_i \in \mathbb{N}^*$, $\sim \in \{<, \leq, =, \geq, >\}$ and $\psi_i \in \mathsf{CLTL}$.

In CLTL formulas, we make use of the standard abbreviations $\vee$, $\Rightarrow$, $\Leftrightarrow$, $\bot$, $\top$, as well as the additional modality $\mathsf{F}_{[C]}\varphi \stackrel{\text{def}}{=} \top \mathsf{U}_{[C]} \varphi$, and its dual $\mathsf{G}_{[C]}\varphi \stackrel{\text{def}}{=} \neg \mathsf{F}_{[C]} \neg\varphi$. Moreover the classical Next operator X is defined as $\mathsf{F}_{[\sharp\top=1]}$, the standard *Until* U is $\mathsf{U}_{[\top]}$ (F is $\mathsf{F}_{[\top]}$ and G is $\mathsf{G}_{[\top]}$). Any formula occurring in a constraint C associated with a modality in Φ is considered as a sub-formula of Φ. The size $|\Phi|$ of Φ takes the size of these constraints and their sub-formulas into account, assuming that integer constants are encoded in *binary* (unless explicitly stated otherwise). The DAG-size of Φ is the total number of *distinct* sub-formulas of Φ. As model-checking algorithms compute only once the truth value of a sub-formula, this is generally more relevant to the complexity of model-checking.

The semantics of CLTL formulas is defined over infinite words in $(2^{\mathsf{AP}})^\omega$:

Definition 3. The following clauses define the conditions for an infinite word $w \in (2^{\mathsf{AP}})^\omega$ to satisfy a CLTL formula φ – written $w \models \varphi$ – by induction over the structure of φ:

$$\begin{array}{ll}
w \models P & \text{iff } P \in w_0 \\
w \models \varphi \wedge \psi & \text{iff } w \models \varphi \text{ and } w \models \psi \\
w \models \neg\psi & \text{iff } w \not\models \psi \\
w \models \varphi \mathsf{U}_{[C]} \psi & \text{iff } \exists i \geq 0, w^i \models \psi,\ w, i-1 \models C \\
 & \qquad \text{and } \forall 0 \leq j < i,\ w^j \models \varphi
\end{array}$$

The semantics of $w, i \models C$ is based on the interpretation of $\sharp\varphi$ over the suffixes w^j for $0 \leq j \leq i$, denoted by $|w, i|_\varphi$ and defined as: $|w, i|_\varphi \stackrel{\text{def}}{=} |\{j \mid 0 \leq j \leq i \wedge w^j \models \varphi\}|$. Given these values, C is interpreted in a natural way (and $\top$ is true over every word).

Given a KS $\mathcal{S} = \langle Q, q_{\text{init}}, R, \ell \rangle$, we write $\mathcal{S} \models \Phi$ when every execution $\rho \in \mathsf{Runs}(\mathcal{S})$ satisfies Φ (*i.e.* $\ell(\rho(0)) \cdot \ell(\rho(1)) \cdots \models \Phi$). We use $\equiv$ to denote the standard equivalence between formulas.

Remark 1. Let us denote by $\overline{C}$ the constraint dual to C obtained by propagating the negation operator in $\neg C$ towards

atomic constraints (using De Morgan's laws and inverting comparison operators as required).

Negation and disjunction operators can be eliminated from constraints using the fact that $\varphi \mathsf{U}_{[\neg C]}\psi \equiv \varphi \mathsf{U}_{[\overline{C}]}\psi$ and $\varphi \mathsf{U}_{[C \vee C']}\psi \equiv \varphi \mathsf{U}_{[C]}\psi \vee \varphi \mathsf{U}_{[C']}\psi$. However, even though $\varphi \mathsf{U}_{[C \wedge C']}\psi \Rightarrow \varphi \mathsf{U}_{[C]}\psi \wedge \varphi \mathsf{U}_{[C']}\psi$, the converse does not hold, as can be seen on the simple example $\mathsf{F}_{[\sharp P_1 = 1 \wedge \sharp P_2 = 1]}\top$ (indeed this formula requires that at some point *both* P_1 and P_2 must have been seen exactly once, while $\mathsf{F}_{[\sharp P_1 = 1]}\top \wedge \mathsf{F}_{[\sharp P_2 = 1]}\top$ does not: for instance P_1 may occur twice before P_2 first occurs).

This implies that any CLTL formula can be translated into an equivalent formula where all constraints are of the form $\bigwedge_i \alpha_i \cdot \sharp\varphi_i \sim k$. However, this may yield an exponentially longer formula, since it essentially requires constraints to be put into disjunctive normal form.

Manipulating constraints. We now define two operations on constraints, which will play an important technical role in the remainder of the paper.

Let C be a counting constraint containing m atomic constraints ($m > 0$) of the form $\sum_{j \in [1,n_i]} \alpha_j^i \cdot \sharp\varphi_j^i \sim k_i$ for $i \in [1,m]$. We define S_C as the set $\{\varphi_j^i \mid i \in [1,m], j \in [1,n_i]\}$. For any $\Delta \subseteq S_C$, we inductively define the subtractive update $C - \Delta$ of C by Δ by:

$$\begin{aligned}
\neg C - \Delta &\stackrel{\text{def}}{=} \neg(C - \Delta) \\
(C \wedge C') - \Delta &\stackrel{\text{def}}{=} (C - \Delta) \wedge (C' - \Delta) \\
\left(\textstyle\sum_i \alpha_i \cdot \sharp\varphi_i \sim k\right) - \Delta &\stackrel{\text{def}}{=} \textstyle\sum_i \alpha_i \cdot \sharp\varphi_i \sim k' \\
&\qquad \text{with } k' \stackrel{\text{def}}{=} k - \textstyle\sum_{\varphi_j \in \Delta} \alpha_j.
\end{aligned}$$

Notice that even though constants in C are defined to be positive integers, $C - \Delta$ may contain negative constants as right-hand sides of comparison operators. However, it can easily be seen that atomic constraints where negative constants (or possibly 0) occur are either trivially true or trivially false. We thus define a second update operation, called *simplification*.

We define the constraint $C{\downarrow}$ obtained from C by replacing any (trivially true) atomic constraint of the form $S > k$ with $k < 0$ or $S \geq k$ with $k \leq 0$ by $\top$ (where S stands for an arbitrary sum of counting expressions), and any (trivially false) atomic constraint of the form $S < k$ with $k \leq 0$ or $S \leq k$ with $k < 0$ by $\bot$, and simplifying the obtained constraint in the usual way (as one would simplify a propositional logic formula). Note that $C{\downarrow}$ is either reduced to $\top$ or $\bot$, or does not contain $\top$ or $\bot$ as a sub-formula. Also note that C and $C{\downarrow}$ are equivalent.

We will write $C' \sqsubseteq C$ whenever there exists a set $\Delta \subseteq S_C$ such that $C' = (C - \Delta){\downarrow}$, and $C' \sqsubset C$ if $\Delta \neq \varnothing$. This notation is extended to CLTL formulas in a natural way ($\varphi \mathsf{U}_{[C']}\psi \sqsubseteq \varphi \mathsf{U}_{[C]}\psi$ if $C' \sqsubseteq C$). It can be shown that $\sqsubset$ is a well-founded strict partial ordering over CLTL formulas.

3. Expressivity

Unfolding. In classical LTL, a crucial observation is that formula $\varphi_1 \mathsf{U} \varphi_2$ can be "unfolded" by distinguishing the possible cases in the first state of a run, yielding the following equivalence:

$$\varphi_1 \mathsf{U} \varphi_2 \equiv \varphi_2 \vee \Big(\varphi_1 \wedge \mathsf{X}(\varphi_1 \mathsf{U} \varphi_2)\Big)$$

In order to obtain a similar equivalence for a formula $\varphi_1 \mathsf{U}_{[C]} \varphi_2$ in counting LTL we need to take into account all the counting expressions occurring in C, and to update the relevant atomic constraints accordingly. To this end we make use of the two elementary update operations on constraints defined in the previous section.

Lemma 1. *For all word w in $(2^{AP})^\omega$ and index $i \geq 0$, $w, i \models C \iff w^1, i-1 \models (C - \Delta){\downarrow}$, where $\Delta = \{\varphi \in S_C \mid w \models \varphi\}$.*

Proof. Let $\sum_i \alpha_i.\sharp\varphi_i \sim k$ be any atomic constraint in C, and $\sum_i \alpha_i.\sharp\varphi_i \sim k'$ with $k' = k - \sum_{\varphi_j \in \Delta} \alpha_j$ the corresponding constraint in $C' = C - \Delta$. By definition of Δ, for every $\varphi \in \Delta$ we have

$$\begin{aligned}
|w,i|_\varphi &= |\{j \mid 0 \leq j \leq i \wedge w^j \models \varphi\}| \\
&= 1 + |\{j \mid 0 < j \leq i \wedge w^j \models \varphi\}| \\
&= 1 + |\{j \mid 0 \leq j \leq i-1 \wedge w^{1+j} \models \varphi\}| \\
&= 1 + |w^1, i-1|_\varphi.
\end{aligned}$$

Similarly for every $\varphi \notin \Delta$, $|w,i|_\varphi = |w^1, i-1|_\varphi$. Thus

$$\begin{aligned}
k - \textstyle\sum_i \alpha_i.|w,i|_\varphi &= k - \textstyle\sum_{\varphi_j \in \Delta} \alpha_j - \sum_i \alpha_i.|w^1, i-1|_{\varphi_i} \\
&= k' - \textstyle\sum_i \alpha_i.|w^1, i-1|_{\varphi_i}.
\end{aligned}$$

Since every atomic constraint of C is satisfied over w at position i if and only if the corresponding constraint in C' is satisfied over w^1 at position $i-1$, and C and C' have otherwise identical structures in terms of Boolean combinations, we get that $w, i \models C \iff w^1, i-1 \models C - \Delta$, which entails the result since the simplification operation does not change the validity of a constraint. □

This enables us to express the effect of the first step in a run on a formula's constraints. We can now come up with an unfolding property similar to LTL. The intuitive idea is to guess the subset $\Gamma \subseteq \mathsf{S}_C$ of formulas accounted for in constraint C which hold over the word at position 0, check that this guess is correct and update C accordingly as described in the previous lemma.

Proposition 2 (Unfolding)**.** *Let* $\Phi = \varphi_1 \mathsf{U}_{[C]} \varphi_2$ *and*

$$\Psi = \bigvee_{\Gamma \subseteq S_C} \big(\bigwedge_{\psi \in \Gamma} \psi \wedge \bigwedge_{\psi \in S_C \setminus \Gamma} \neg\psi \wedge \varphi_1 \wedge \mathsf{X}(\varphi_1 \mathsf{U}_{[(C-\Gamma)\downarrow]} \varphi_2) \big).$$

The following equivalence holds:

$$\Phi \equiv \begin{cases} \Psi \vee \varphi_2 & \text{if } w, -1 \models C, \\ \Psi & \text{otherwise.} \end{cases}$$

Proof. $\Phi \Rightarrow \Psi / \Psi \vee \varphi_2$: If Φ is satisfied over some word $w \in (2^{\mathsf{AP}})^\omega$, then by definition $\exists i \geq 0$, $w^i \models \varphi_2$, $w, i{-}1 \models C$ and $\forall 0 \leq j < i$, $w^j \models \varphi$.

If $i = 0$, i.e. $w, -1 \models C$ and $w \models \varphi_2$, then $\Psi \vee \varphi_2$ holds. Otherwise ($i > 0$) it must be that $w \models \varphi_1$ and $w, i{-}1 \models C$. Let Δ be the set of formulas of S_C which hold over w, by Lemma 1 we have $w^1, i-2 \models (C - \Delta)\downarrow$. Furthermore there exists a disjunct in Ψ (namely when $\Gamma = \Delta$) such that $\bigwedge_{\psi \in \Gamma} \psi \wedge \bigwedge_{\psi \in S_C \setminus \Gamma} \neg\psi$ holds. Finally, we can deduce from all of the above that $(w^1)^{i-1} \models \varphi_2$, $w^1, i-2 \models (C - \Delta)\downarrow$ and $\forall 0 \leq j < i - 1$, $(w^1)^j \models \varphi$, in other words $w^1 \models \varphi_1 \mathsf{U}_{[(C-\Delta)\downarrow]} \varphi_2$. Together with the above observations, this implies that $w \models \Psi$.

$\Psi \Rightarrow \Phi$: Let $w \models \Psi$, there must exist Γ such that $w \models \bigwedge_{\psi \in \Gamma} \psi \wedge \bigwedge_{\psi \in S_C \setminus \Gamma} \neg\psi \wedge \varphi_1 \wedge \mathsf{X}(\varphi_1 \mathsf{U}_{[(C-\Gamma)\downarrow]} \varphi_2)$. From this, we can deduce that (1) $\exists i > 0, w, i+1 \models \varphi_2$ and $\forall 0 \leq j \leq i, w, j \models \varphi_1$, (2) $w^1, i \models (C - \Gamma)\downarrow$ and (3) $\Gamma = \{\varphi \in S_C \mid w \models \varphi\}$ which by Lemma 1 entails that $w, i+1 \models C$. Together with item (1) above, we get that $w \models \Phi$. □

Remark 2. Note that even a single unfolding step as described by the previous proposition may entail an exponential increase in the dag-size of the formula, since the set Γ needs to be guessed explicitly. This blow-up can be kept polynomial by "scanning" formulas in S_C one at a time and in a fixed order instead of considering all possible $\Gamma \subseteq S_C$. This technique was used in [12] to study the translation of a fragment of CCTL into CTL.

Expressivity and succinctness. Similarly to the corresponding counting CTL logic [12], CLTL is not more expressive than classical LTL.

Proposition 3 (Expressivity)**.** *Any CLTL formula can be translated into LTL.*

Proof. We reason by induction on the structure of Φ. The case of Boolean connectives is trivial. We treat the case $\Phi = \varphi \mathsf{U}_{[C]} \psi$ by induction on the well-founded partial ordering $\sqsubset$ defined in the previous section.

If Φ is minimal for $\sqsubset$ (i.e. $C \in \{\top, \bot\}$), we can directly use the inductive LTL translations of φ and ψ, since $\varphi \mathsf{U}_{[\bot]} \psi \equiv \bot$ and $\varphi \mathsf{U}_{[\top]} \psi \equiv \varphi \mathsf{U} \psi$.

Next, if $C \notin \{\top, \bot\}$, it is easy to show that

$$\begin{aligned} \Phi &\equiv (\textstyle\bigwedge_{\psi \in S_C} \neg\psi) \mathsf{U} \big((\bigvee_{\psi \in S_C} \psi) \wedge (\varphi_1 \mathsf{U}_{[C]} \varphi_2) \big) \\ &\equiv (\textstyle\bigwedge_{\psi \in S_C} \neg\psi) \mathsf{U} \big((\bigvee_{\psi \in S_C} \psi) \wedge \Phi' \big) \end{aligned}$$

where Φ' is $\Psi' \vee \varphi_2$ if $w, -1 \models C$ and Ψ' otherwise, and Ψ' is identical to formula Ψ in Prop. 2 above, omitting the disjunct for $\Gamma = \varnothing$. Now the top-most constraints C' occurring in Ψ' are equal to $(C - \Gamma)\downarrow$ with some non-empty Γ, and thus $\Psi' \sqsubset \Phi$. By induction hypothesis, Ψ' can be translated into LTL, which concludes the proof. □

However, this translation may yield an exponential increase in dag-size, since the number of distinct constraints $C' \sqsubset C$ is of the order of M^m (with M the largest constant and m the number of atomic constraints occurring in C), hence also in $2^{O(|\Phi|^2)}$. We are as of yet not able to show that this bound is tight, but there exist CLTL formulas whose shortest equivalent LTL formula is provably of dag-size at least in $O(M)$.

Proposition 4 (Succinctness)**.** *Any LTL formula equivalent to the CLTL formula* $\Phi_k = \mathsf{F}(\neg b \mathsf{U}_{[\sharp a = k]} \top)$ *has temporal depth at least* $k - 1$ *(i.e. exponential in* $|\Phi_k|$*).*

Proof. Consider the set $\mathsf{AP} = \{a, b, c\}$, and the property $\textsc{Stairs}_k$ ([9]), which states that there exists a portion of the path in which proposition a occurs at least k times but proposition b does not occur. In [9], it is shown that this property can only be expressed by a LTL formula with at least $k - 1$ nested *Until* modalities. However, this formula is equivalent to the CLTL formula Φ_k. □

4. Decision procedures

We consider two standard decision problems for CLTL, namely satisfiability (given $\Phi \in$ CLTL, does there exists a model for Φ ?) and model checking (given $\Phi \in$ CLTL and some KS $\mathcal{S}$, do all runs of $\mathcal{S}$ satisfy Φ, *i.e.* $\mathcal{S} \models \Phi$?).

Classical decision procedures for LTL satisfiability are based on automata constructions. Given some LTL formula Φ, one can either build an (exponential) non-deterministic Büchi automaton or a (polynomial) alternating Büchi automaton accepting exactly the models of Φ. Satisfiability then consists in checking whether the language of the automaton is empty [15]. We begin this section by recalling the definition of alternating Büchi automata, then extend the usual automata-based decision procedures for satisfiability and model-checking to our logic CLTL.

4.1. Alternating Büchi Automata over ω-words

An *alternating* Büchi automaton on infinite words is a tuple $\mathcal{A} = (\Sigma, S, s^0, \delta, F)$ where Σ is a finite alphabet, S

is a finite set of states, $s^0 \in S$ is the initial state, $\delta : S \times \Sigma \to \mathcal{B}^+(S)$ is the transition function assigning a positive Boolean formula over S (including $\bot$ and $\top$) to every pair (s, σ), and $F \subseteq S$ is the Büchi acceptance condition.

A run over an infinite word $w = a_0 a_1 \cdots \in \Sigma^\omega$ is an infinite S-labeled tree $\mathcal{T} = (T, l)$ where T is a tree and $l : \text{Nodes}(T) \to S$ assigns an element in S to every node in T. The root ϵ of T has to be labeled by s^0 (*i.e.* $l(\epsilon) = s^0$) and every node x at depth i (written $|x| = i$) has k ($k \geq 0$) children x_1,...,x_k such that the formula $\delta(l(x), a_i)$ is interpreted to true when one assigns $\top$ to every state in $\{l(x_1), \ldots, l(x_k)\}$ and $\bot$ to other states.

The run is accepted when every infinite branch of $\mathcal{T}$ contains infinitely often nodes labeled by states in F and every finite branch ends in a node x such that $\delta(l(x), a_{|x|}) = \top$. We use $\mathcal{L}(\mathcal{A})$ to denote the set of words accepted by $\mathcal{A}$.

4.2. Satisfiability

By using the standard techniques for LTL, one obtains the following results:

Proposition 5. *Given a CLTL formula Φ, one can build an alternating Büchi automaton $\mathcal{A}_\Phi$ such that (1) $|\mathcal{A}_\Phi|$ is in $O(|\Phi| \cdot M^{|\Phi|})$ where M is the maximal constant occurring in constraints inside Φ, and (2) $L_\omega(\mathcal{A}_\Phi)$ is exactly the set of runs satisfying Φ.*

Proof. Let Φ be a CLTL formula. Let M be the maximal constant occurring in the counting constraints in Φ and m the maximal number of atomic constraints $\sum_i \alpha_i \cdot \sharp\psi_i \sim k$ occurring in the same constraint in Φ.

We define $\mathcal{A}_\Phi = (\Sigma, S_\Phi, s^0, \delta, F)$, where Σ is 2^{AP}, S_Φ is the set of all subformulas of Φ (including those appearing in constraints), $\varphi_1 \mathsf{U}_{[(C-\Delta)\downarrow]} \varphi_2$ for every subformula $\varphi_1 \mathsf{U}_{[C]} \varphi_2$ and $\Delta \subseteq S_C$, and their negations, s^0 is Φ, $\delta : S_\Phi \times \Sigma \to \mathcal{B}^+(S_\Phi)$ is the transition function defined below and F contains every state in S of the form $\neg(\varphi_1 \mathsf{U} \varphi_2)$ or $\neg(\varphi_1 \mathsf{U}_{[C]} \varphi_2)$.

In the following we use $\overline{\theta}$ to denote the negation normal form of the formula $\theta \in \mathcal{B}^+(S_\Phi)$: every conjunction (resp. disjunction) becomes a disjunction (resp. conjunction), $\top$ (resp. $\bot$) becomes $\bot$ (resp. $\top$), and $\overline{\overline{\theta}}$ is just θ. Negated states are fine since $\varphi \in S_\Phi \Rightarrow \neg\varphi \in S_\Phi$.

For convenience, we define of the transition function recursively. Occurrences of $\delta(\varphi, \sigma)$ in right-hand sides should be replaced by their definition until a formula in $\mathcal{B}^+(S_\Phi)$ is obtained. We have $\delta(P, \sigma) = \top$ if $P \in \sigma$ and $\bot$ otherwise, $\delta(\varphi \wedge \psi, \sigma) = \delta(\varphi, \sigma) \wedge \delta(\psi, \sigma)$, and $\delta(\neg\varphi, \sigma) = \overline{\delta(\varphi, \sigma)}$. The rule for U is based on the unfolding rule (see Prop. 2): $\delta(\varphi_1 \mathsf{U}_{[C]} \varphi_2, \sigma) = \delta(\varphi_2, \sigma) \vee \theta$ if $\rho, -1 \models C$ and θ otherwise, with

$$\theta = \bigvee_{\Gamma \subseteq S_C} \big(\bigwedge_{\psi \in \Gamma} \delta(\psi, \sigma) \wedge \bigwedge_{\psi \in S_C \setminus \Gamma} \delta(\neg\psi, \sigma) \wedge \delta(\varphi_1, \sigma) \wedge (\varphi_1 \mathsf{U}_{[(C-\Gamma)\downarrow]} \varphi_2) \big).$$

The number of states is in $O(|\Phi| \cdot M^m)$: every $\varphi_1 \mathsf{U}_{[C]} \varphi_2$ subformula may provide $(M+2)^m$ states. Also note that the transition formula θ above can be expressed in a more concise way using a more refined unfolding technique (Cf. Rem. 2), at the cost of roughly duplicating $|S_C|$ times the states corresponding to each $\mathsf{U}_{[C]}$-subformula. This automaton recognizes exactly the models of Φ. □

The complexity of this algorithm is in fact asymptotically optimal:

Theorem 6. *CLTL satisfiability is* EXPSPACE*-complete.*

Proof. Membership in EXPSPACE is based on Prop. 5: the size of the automaton $\mathcal{A}_\Phi$ is in $2^{O(|\Phi|^2)}$ and checking emptiness of an alternating Büchi automaton is PSPACE-complete [5]. This provides an EXPSPACE algorithm.

First note that EXPSPACE-hardness is a consequence of the complexity of TLTL (*i.e.* Timed LTL) over discrete time domains [11]. Nevertheless we give a proof based on the encoding in CLTL of the execution of a Turing Machine running in exponential space over some input word (such an encoding is classical, see for example [3]).

Consider a deterministic 2^n-space-bounded Turing machine $\mathcal{M} = \langle \Sigma, Q_\mathcal{M}, q_0, q_F, R_\mathcal{M} \rangle$, with an initial tape content $X = x_1 \ldots x_n$. We assume w.l.o.g. $\Sigma = \{a, b\}$. q_0 is the initial state and q_F is the final state. And as usual $R_\mathcal{M} \subseteq Q_\mathcal{M} \times \Sigma \times \Sigma \times \{-1, 1\} \times Q_\mathcal{M}$.

Now we construct a polynomial-size formula describing the accepting computation of $\mathcal{M}$ on X. The set of atomic propositions AP is defined as follows: AP contains P_a and P_b to represent the corresponding symbol on the tape, an additional proposition P_s to separate two consecutive configurations, and propositions $P_{a,q}$ and $P_{b,q}$ for every $q \in Q_\mathcal{M}$ to mark the position of the tape head on a cell containing a symbol a or b respectively.

A configuration of $\mathcal{M}$ is encoded as a sequence of 2^n states labeled with propositions in AP to represent the content of the cells. One of these cell is labeled with some $P_{a,q}$ or $P_{b,q}$, and the sequence is preceded and followed by a state labeled with P_s.

In the following we use the abbreviation $P_\varnothing$ to represent $\bigwedge_{P \in \mathsf{AP}} \neg P$. This formula is used to represent empty cells.

To specify that the run is the correct and accepting one, we need a formula of the form $(\Phi_i \wedge \Phi_m) \Rightarrow \Phi_a$ (where i, m and a stand for *init*, *move* and *accept* respectively), meaning that if the run starts with the initial configuration and follows the transitions of $\mathcal{M}$, then it is accepting. These

three formulas can be expressed in CLTL:

$$\begin{aligned}\Phi_i &= P_s \wedge \mathsf{X}(P_{x_1,q_0} \wedge \bigwedge_{2\le k\le n} \mathsf{F}_{[\sharp\top=k]} P_{x_k} \\ &\quad \wedge \mathsf{F}_{[\sharp\top=n+1]}(P_{\varnothing}\mathsf{U}_{[\sharp\top=2^n-n]} P_s) \\ \Phi_m &= \mathsf{G}\big(P_s \Rightarrow \mathsf{X}(\neg P_s)\mathsf{U}_{[\sharp\top=2^n]} P_s\big) \\ &\quad \wedge \bigwedge_{(P_1,P_2,P_3)\in \mathsf{AP}^3} \mathsf{G}\big((P_1 \wedge \mathsf{X}P_2 \wedge \mathsf{XX}P_3) \\ &\quad \Rightarrow \mathsf{F}_{[\sharp\top=2^n+2]} f_{\mathcal{M}}(P_1,P_2,P_3)\big) \\ \Phi_a &= \mathsf{F}(P_{a,q_F} \vee P_{b,q_F}),\end{aligned}$$

where the function $f_{\mathcal{M}}(P_1, P_2, P_3)$ refers to the transition rules of $\mathcal{M}$: $f_{\mathcal{M}}(P_1, P_2, P_3)$ gives the value of the cell containing P_2 in the next configuration given the definition of the left cell (P_1) and the right cell (P_3). For instance, for every rule $(q, a, b, +1, q')$ in $R_{\mathcal{M}}$ we will have: $f_{\mathcal{M}}(P_1, P_{a,q}, P_2) = P_b$ for any $P_1 \in \mathsf{AP}$ and any $P_2 \neq P_s$. Moreover we have for any $P_1 \in \mathsf{AP}$, the two values: $f_{\mathcal{M}}(P_{a,q}, P_a, P_1) = P_{a,q'}$ and $f_{\mathcal{M}}(P_{a,q}, P_b, P_1) = P_{b,q'}$. And we also define $f_{\mathcal{M}}(P_1, P_2, P_3) = P_2$ if neither P_1 or P_3 are of the form $P_{a,q}$ or $P_{b,q}$ for some q.

The lengths of formulas Φ_i, Φ_m and Φ_a are polynomial, since constants are encoded in binary, which implies the EXPSPACE-hardness of CLTL satisfiability. □

Note that if constraints are atomic (*i.e.* without Boolean combinations in subscripts), then m is equal to 1 and the size of $\mathcal{A}_\Phi$ is in $O(|\Phi| \cdot M)$. If in addition, constants are assumed to be encoded in unary, the satisfiability algorithm becomes PSPACE.

4.3. Model-checking

Corollary 1. *The model-checking problem for CLTL is* EXPSPACE*-complete.*

Proof. Hardness for EXPSPACE comes from that of satisfiability, which can be reduced to a model-checking problem using some kind of *universal* Kripke structure $\mathcal{S}_u$ able to generate any possible word in $(2^{\mathsf{AP}})^\omega$: Φ is satisfiable iff $\mathcal{S}_u \not\models \neg\Phi$. Let AP be $\{P_1, \ldots, P_n\}$. Instead of considering a complete KS whose states are labeled with every possible subset of AP (which would yield an exponential structure), we use a succinct KS $\mathcal{S}'_u$ that encodes every valuation of a state in $\mathcal{S}_u$ as a sequence of n states labeled respectively by $\varnothing$ or P_i... It then remains to slighthly modify Φ to take into account this encoding. Let Φ' be the modified formula, we can reduce $\mathcal{S}_u \models \Phi$ to $\mathcal{S}'_u \models \Phi'$.

Membership in EXPSPACE is obtained following the idea for classical LTL model-checking. Given a Kripke Structure S and a CLTL formula Φ, one builds as previously an alternating Büchi automaton $\mathcal{A}$ for the formula $\neg\Phi$. It is then straightforward to compute the product of $\mathcal{A}$ with the structure S in such a way that the obtained automaton has an accepting infinite run if and only if there exists a path in S violating Φ. □

Note that the program complexity of model-checking for CLTL (i.e. the complexity of model-checking a *fixed* formula) is (like for LTL) NL-complete [16].

4.4. A PSPACE fragment of CLTL

The EXPSPACE-hardness proof of CLTL satisfiability only uses counting constraints of the form "$\sharp\top = k$": there is no need for nested formulas in constraints, no Boolean combinations and no sums. Here we introduce the fragment CLTL^- defined as the set of CLTL formulas where counting constraints are purely conjunctive terms, and comparison symbols are not mixed inside a constraint. In other terms, constraints are of the form "$\bigwedge \sum_i \alpha_i \cdot \sharp\psi_i < k$", "$\bigwedge \sum_i \alpha_i \cdot \sharp\psi_i > k$" or their non-strict variants. Note that this restriction also applies over subformulas in constraints.

We use $\varphi_1 \mathsf{U}_{[C\prec]}\varphi_2$ (resp. $\varphi_1 \mathsf{U}_{[C\succ]}\varphi_2$) to denote an *Until*-subformula tagged with a constraint of the form "less than" *i.e.* with $\leq$ or $<$ (resp. "greater than" with $\geq$ or $>$).

In the following theorem, we claim that CLTL^- formulas admit PSPACE decision procedures:

Theorem 7. *The satisfiability and model-checking problems for* CLTL^- *are* PSPACE*-complete.*

Proof. PSPACE-hardness comes from LTL satisfiability. PSPACE membership is based on the fact that given a CLTL^- formula Φ and $\mathcal{A}_\Phi$ the corresponding automaton as built in Proposition 5, for any accepting run over some model w of Φ, there exists a "small" accepting run over w. By small, we mean a tree with a *width* (*i.e.* the maximal number of nodes at the same level) bounded by $|\Phi|$.

Let Φ be a CLTL^- formula. First we can assume that Φ only contains atomic constraints (with no conjunction): indeed every CLTL^- formula $\varphi_1 \mathsf{U}_{[C\wedge C']}\varphi_2$ is equivalent to $\varphi_1 \mathsf{U}_{[C]}\varphi_2 \wedge \varphi_1 \mathsf{U}_{[C']}\varphi_2$. This translation can be done efficiently and the dag-size of the resulting formula is linear in the size of the original one. Let *Subf*(Φ) be the set of subformulas of Φ.

Now consider $\mathcal{A}_\Phi$ as defined in Proposition 5. The number of states of $\mathcal{A}_\Phi$ is in $O(|\Phi| \cdot M)$ where M is the size of the maximal constant occurring in Φ. Thus this number is exponential in $|\Phi|$ (this blow-up is due to the rewriting of $\varphi_1 \mathsf{U}_{[C]}\varphi_2$ subformulas into $\varphi_1 \mathsf{U}_{[C-\Gamma]}\varphi_2$ subformulas in the function δ).

Now consider an accepting run $\mathcal{T} = (T, l)$ of $\mathcal{A}_\Phi$ over an infinite word w that is a model of Φ. At every level i of the tree T, the nodes $\{x_1, \ldots, x_k\}$ are labeled with the set of formulas $\{l(x_1), \ldots, l(x_k)\} \subseteq S_\Phi$ (see the definition of S_Φ in Prop. 5) and every formula $l(x_j)$ holds over the word w^i. For every $\psi \in$ *Subf*(Φ) of the form $\varphi_1 \mathsf{U}_{[C]}\varphi_2$,

it is possible to have several formulas $\varphi_1 \mathsf{U}_{[C-\Gamma]} \varphi_2$ for different subsets Γ of S_C. But we clearly only need to verify one formula of this set: if ψ is a "less than" (resp. a "greater than") formula, we consider the one containing the minimal (resp. maximal) constant k in the constraint. Indeed we clearly have $\varphi_1 \mathsf{U}_{[C<k]} \varphi_2 \Rightarrow \varphi_1 \mathsf{U}_{[C<k']} \varphi_2$ for any $k \leq k'$ and $\varphi_1 \mathsf{U}_{[C>k]} \varphi_2 \Rightarrow \varphi_1 \mathsf{U}_{[C>k']} \varphi_2$ for any $k \geq k'$.

Then at every level of the tree, we only need to keep one formula among this subset of formulas $\{\varphi_1 \mathsf{U}_{[C-\Gamma]} \varphi_2 \mid \Gamma \subseteq S_C\}$. Thus we can ensure the number of formulas labeling states at some level to be bounded by $|\Phi|$. This remark leads to an NSPACE algorithm for satisfiability (and model checking). It works as follows.

Let S_i be the set of S_Φ formulas labeling states of level i: we have $|S_i| \leq |\Phi|$ and this set can be encoded in polynomial space (w.r.t. $|\Phi|$). Now the procedure guesses nondeterministically a letter w_i and a subset S_{i+1} and verifies that it may correspond to the level $i+1$. For this, the algorithm has to check $S_{i+1} \models \delta(\psi, w_i)$ for every $\psi \in S_i$: this is done again with a non-deterministic choice of subsets Γ in the function δ and by interpreting $\varphi_1 \mathsf{U}_{[C\prec]} \varphi_2$ (resp. $\varphi_1 \mathsf{U}_{[C\succ]} \varphi_2$) as true if there is some formula $\varphi_1 \mathsf{U}_{[C-\Gamma]} \varphi_2$ in S_{i+1} (resp. $\varphi_1 \mathsf{U}_{[C']} \varphi_2$ in S_{i+1} with $C = C' - \Gamma$).

Moreover as usual for this kind of algorithms, the procedure will guess non-deterministically that some level ℓ is the first state of a cycle and will verify that there is a future level labeled with the same set of formulas S_ℓ: to do this we simply need to memorize S_ℓ.

Finally we need to verify that the acceptance condition is satisfied by the final cycle from level ℓ. This is done by checking that every formula $\varphi_1 \mathsf{U}_{[C]} \varphi_2 \in S_\ell$ is satisfied somewhere along the cycle (there must be no branch along which the label $\varphi_1 \mathsf{U}_{[C]} \varphi_2$ ultimately appears forever). For this, we need to store (and update) the *Until*-subformulas that have not yet been satisfied along the cycle, and mark each of them as soon as the corresponding φ_2 holds, which can be done step by step by analyzing the function δ. Once the set of formulas S_ℓ is repeated, we need to have successfully asserted this fact for every formula (or one of its descendants with constraint $C - \Gamma$). Note that every $\varphi_1 \mathsf{U}_{[C]} \varphi_2$ that does not occur at level ℓ but appears inside the cycle will be either satisfied before the next occurrence of S_ℓ, or will yield a subformula $\varphi_1 \mathsf{U}_{[C']} \varphi_2$ in S_ℓ and then will be treated as in the previous case.

This yields an NSPACE procedure and by Savitch's theorem one can deduce the existence of a PSPACE algorithm. The model-checking algorithm is based on the same technique for analyzing the alternating automaton. □

This result is another illustration of the potential complexity cost of equality in quantitative constraints as in the timed case [1].

5. Extension with diagonal constraints

In [12], we presented several decidable fragments of CCTL in which *atomic* constraints with subtraction were allowed. In this section, we show that even a simple extension of LTL with such constraints leads to undecidability. More formally, we consider the logic obtained from CLTL by replacing the constraint language $\mathcal{C}$ with the language $\mathcal{C}'$ of constraints of the form $\sharp\varphi_1 - \sharp\varphi_2 \sim k$ (*i.e.* with no Boolean combination), which we call *diagonal* constraints. It turns out that, unlike CCTL where model-checking remains polynomial for this restricted case, this constraint language yields undecidability in the case of CLTL.

Theorem 8. *The model-checking and satisfiability problems for CLTL with atomic diagonal constraints are undecidable.*

Proof. This is done by reduction from the halting problem of a two-counter machine $\mathcal{M}$ with counters C and D, and n instructions $I_1, \ldots, I_n$. Each I_i is either a decrement $\langle \texttt{if } X\texttt{=0 then } j \texttt{ else } X\texttt{--, } k\rangle$ where X stands for C or D, an increment $\langle X\texttt{++, } j\rangle$, or the halting instruction $\langle \texttt{halt}\rangle$. We define a Kripke structure $\mathcal{S}_\mathcal{M} = (Q, R, \ell)$, where $Q = \{q_1, \ldots, q_n\} \cup \{r_i, t_i \mid I_i = \langle \texttt{if } \ldots\rangle\}$. The transition relation is defined as follows:

- if $I_i = \langle X\texttt{++, } j\rangle$, then $(q_i, q_j) \in R$; and
- if $I_i = \langle \texttt{if } X\texttt{=0 then } j \texttt{ else } X\texttt{--,} k\rangle$, then (q_i, r_i), (r_i, q_k), (q_i, t_i) and (t_i, q_j) in R.

The labeling ℓ is defined over the set $\{\mathsf{halt}, C^+, C^-, C^0, D^+, D^-, D^0\}$ as $\ell(q_i) = \{X^+\}$ if I_i is an increment of X, $\ell(r_i) = \{X^-\}$ and $\ell(t_i) = \{X^0\}$ if I_i is a decrement for X, and $\ell(q_i) = \{\mathsf{halt}\}$ if I_i is the halting instruction.

A run going through t_i for some i will simulate the positive test "$X = 0$": we use the proposition X^0 to observe this fact. Indeed along any run in $\mathcal{S}_\mathcal{M}$, a state satisfies X^0 if and only if that state is some t_i state, which witnesses the fact that the counter's value was deemed equal to zero. The propositions on the other states are self-explanatory, witnessing increments and decrements of counters.

Checking CLTL with atomic diagonal constraints on this structure solves the halting problem, since $\mathcal{M}$ halts *if and only if* $\mathcal{S}_\mathcal{M} \models \Phi$ with:

$$\Phi = \mathsf{F}_{[(\sharp\mathsf{halt} \geq 1)]} \top \\ \vee \bigvee_{X \in \{\mathtt{C},\mathtt{D}\}} \left(\mathsf{F}_{[(\sharp X^+ - \sharp X^- < 0)]} \top \vee \mathsf{F}_{[(\sharp X^+ - \sharp X^- > 0)]} X^0 \right)$$

The formula Φ is satisfied by a run because either $\mathcal{S}_\mathcal{M}$ halts, or the run does not simulate correctly $\mathcal{M}$ because the number of decrements is at some point larger than the number of increments, or because some counter was incorrectly assumed to be zero while simulating a test. Thus, if Φ is true for every run, it is in particular the case of the path simulating the behavior of $\mathcal{M}$. □

6. CCTL*

Using similar modalities in a branching framework, one can define a counting extension of the logic CTL*.

Definition 4. Let AP be a set of atomic propositions, we distinguish:

$$\mathsf{CCTL}^* \ni \varphi_s, \psi_s \ ::= \ P \mid \varphi_s \wedge \psi_s \mid \neg\varphi_s \mid \mathsf{E}\varphi_p$$
$$\mathsf{CCTL}^*_{\mathrm{p}} \ni \varphi_p, \psi_p \ ::= \ \varphi_s \mid \varphi_p \wedge \psi_p \mid \neg\varphi_p \mid \varphi_p \mathsf{U}_{[C]} \psi_p$$

where C denotes a counting constraint as in Def. 2 with subformulas in $\mathsf{CCTL}^* \cup \mathsf{CCTL}^*_{\mathrm{p}}$.

The semantics of CCTL* formulas is defined over states of Kripke structures as follows:

Definition 5. The following clauses (Boolean cases are omitted) define the conditions for a state q (resp. a run ρ) of some KS $\mathcal{S} = \langle Q, q_{\text{init}}, R, \ell \rangle$ to satisfy a CCTL* formula φ_s (resp. a CCTL$^*_{\mathrm{p}}$ formula φ_p) by induction over the structure of φ_s (resp. φ_p):

$$\begin{array}{ll}
q \models_{\mathcal{S}} P & \text{iff } P \in \ell(q) \\
q \models_{\mathcal{S}} \mathsf{E}\varphi_p & \text{iff } \exists \rho \in \mathsf{Runs}(q), \rho \models_{\mathcal{S}} \varphi_p \\
\rho \models_{\mathcal{S}} \varphi_s & \text{iff } \rho(0) \models_{\mathcal{S}} \varphi_s \\
\rho \models_{\mathcal{S}} \varphi \mathsf{U}_{[C]} \psi & \text{iff } \exists i \geq 0, \rho^i \models_{\mathcal{S}} \psi,\ \rho, i-1 \models_{\mathcal{S}} C \\
 & \quad \text{and } \forall 0 \leq j < i,\ \rho^j \models_{\mathcal{S}} \varphi
\end{array}$$

We use A to denote the dual of E. The model-checking problem consists in deciding whether a given CCTL* formula holds for a given state in a KS $\mathcal{S}$.

Theorem 9. *The model-checking problem for CCTL* is* EXPSPACE*-complete.*

Proof. EXPSPACE-hardness comes from the corresponding problems for CLTL. EXPSPACE membership is obtained thanks to the EXPSPACE procedure for CLTL formulas. One can design a polynomial-time algorithm that calls an oracle for CLTL subformulas, which provides a $\mathsf{P}^{\mathsf{EXPSPACE}}$ procedure (hence also in EXPSPACE). □

7. Conclusion

We have proposed new extensions for LTL and CTL* which, together with our related results for CTL [12], provide a general overview of expressivity and complexity for a natural class of quantitative temporal logics. There are several possible continuations to this work, some of which we are currently exploring. It would be interesting to evaluate the succinctness and algorithmic properties of the *unary* fragment of CLTL (i.e. CLTL with unary-encoded constants), for which we believe better algorithms may exist despite the fact that it is not clear how to avoid an exponential blow-up in the dag-size of the LTL translation. It would also be natural to consider the addition of *past* modalities, which bring exponential succinctness improvements to LTL with no significant complexity cost. Finally, we are working on different (*cumulative*) semantics for constraints, which evaluate counting expressions over the full history of runs.

References

[1] R. Alur, T. Feder, and T. A. Henzinger. The benefits of relaxing punctuality. *J. ACM*, 43(1):116–146, 1996.

[2] R. Alur and T. A. Henzinger. Logics and models of real time: A survey. In *Proc. REX Workshop*, volume 600 of *LNCS*, pages 74–106. Springer, 1992.

[3] R. Alur and T. A. Henzinger. A really temporal logic. *J. ACM*, 41(1):181–203, 1994.

[4] D. Bustan, D. Fisman, and J. Havlicek. Automata construction for psl. Technical report, The Weizmann Institute of Science, 2005. Available as Tech. Report MCS05- 04.

[5] A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981.

[6] E. A. Emerson. Temporal and modal logic. In *Handbook of Theoretical Computer Science*, volume B, chapter 16, pages 995–1072. Elsevier Science, 1990.

[7] E. A. Emerson, A. K. Mok, A. P. Sistla, and J. Srinivasan. Quantitative temporal reasoning. *Real-Time Systems*, 4(4):331–352, 1992.

[8] E. A. Emerson and R. J. Trefler. Generalized quantitative temporal reasoning: An automata-theoretic approach. In *Proc. 7th TAPSOFT*, volume 1214 of *LNCS*, pages 189–200. Springer, 1997.

[9] K. Etessami and T. Wilke. An until hierarchy and other applications of an Ehrenfeucht-Fraïssé game for temporal logic. *Inf. Comput.*, 160(1-2):88–108, 2000.

[10] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Asp. Comput.*, 6(5):512–535, 1994.

[11] F. Laroussinie, N. Markey, and Ph. Schnoebelen. Efficient timed model checking for discrete-time systems. *Theor. Comput. Sci.*, 353(1-3):249–271, 2006.

[12] F. Laroussinie, A. Meyer, and E. Petonnet. Counting CTL. In *Proc. 13th FoSSaCS*, volume 6014 of *LNCS*, pages 206–220. Springer, 2010.

[13] A. Pnueli. The temporal logic of programs. In *Proc. 18th FOCS*, pages 46–57. IEEE Comp. Soc. Press, 1977.

[14] *Property Specification Language Reference Manual, Version 1.1*, 2003. `http://www.eda-stds.org/vfv/docs/PSL-v1.1.pdf`.

[15] M. Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Logics for Concurrency: Structure Versus Automata*, volume 1043 of *LNCS*, pages 238–266. Springer, 1996.

[16] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. 1st LICS*, pages 332–344. IEEE Comp. Soc. Press, 1986.

[17] P. Wolper. Temporal logic can be more expressive. *Inf. and Control*, 56(1/2):72–99, 1983.