Automatic Generation of Quality Specifications

Shaull Almagor, Guy Avni, and Orna Kupferman

School of Computer Science and Engineering, The Hebrew University, Jerusalem, Israel.

Abstract. The logic $\mathrm{LTL}^{\triangledown}$ extends LTL by quality operators. The satisfaction value of an $\mathrm{LTL}^{\triangledown}$ formula in a computation refines the $0/1$ value of LTL formulas to a real value in $[0,1]$. The higher the value is, the better is the quality of the computation. The quality operator $\triangledown_\lambda$, for a quality constant $\lambda \in [0,1]$, enables the designer to prioritize different satisfaction possibilities. Formally, the satisfaction value of a subformula $\triangledown_\lambda \varphi$ is λ times the satisfaction value of φ. For example, the $\mathrm{LTL}^{\triangledown}$ formula $G(req \rightarrow (X\,grant \vee \triangledown_{\frac{1}{2}} F\,grant))$ has value 1 in computations in which every request is immediately followed by a grant, value $\frac{1}{2}$ if grants to some requests involve a delay, and value 0 if some request is not followed by a grant.

The design of an $\mathrm{LTL}^{\triangledown}$ formula typically starts with an LTL formula on top of which the designer adds the parameterized $\triangledown$ operators. In the Boolean setting, the problem of automatic generation of specifications from binary-tagged computations is of great importance and is a very challenging one. Here we consider the quantitative counterpart: an $\mathrm{LTL}^{\triangledown}$ query is an $\mathrm{LTL}^{\triangledown}$ formula in which some of the quality constants are replaced by variables. Given an $\mathrm{LTL}^{\triangledown}$ query and a set of computations tagged by satisfaction values, the goal is to find an assignment to the variables in the query so that the obtained $\mathrm{LTL}^{\triangledown}$ formula has the given satisfaction values, or, if this is impossible, best approximates them. The motivation to solving $\mathrm{LTL}^{\triangledown}$ queries is that in practice it is easier for a designer to provide desired satisfaction values in representative computations than to come up with quality constants that capture his intuition of good and bad quality.

We study the problem of solving $\mathrm{LTL}^{\triangledown}$ queries and show that while the problem is NP-hard, interesting fragments can be solved in polynomial time. One such fragment is the case of a single tagged computation, which we use for introducing a heuristic for the general case. The polynomial solution is based on an analysis of the search space, showing that reasoning about the infinitely many possible assignments can proceed by reasoning about their partition into finitely many classes. Our experimental results show the effectiveness and favorable outcome of the heuristic.

1 Introduction

Traditional formal methods are based on a Boolean satisfaction notion – a reactive system satisfies, or not, a given specification. In recent years there is growing need and interest in formalizing and reasoning about quantitative systems and properties. This includes, for example, probabilistic [16], fuzzy [18], and accumulative [9] settings. An exciting direction in this effort is the development of formalisms and methods for reasoning about the *quality* of systems [2, 1]. The working assumption in these works is that satisfying a specification is not a yes/no matter. Different ways of satisfying a specification should induce different levels of quality, which should be reflected in the semantics of the specification formalism. In particular, in [2], the authors introduce an extension of linear temporal logic (LTL [19])

by a quantitative layer that enables the designer to prioritize different satisfaction possibilities. In the extended setting, the satisfaction value of a formula in a computation refines the $0/1$ value of LTL formulas to a real value in $[0, 1]$. The higher the value is, the better is the quality of the computation.

The extension uses a family of *propositional quality operators*. A basic such operator is ∇_λ, for a *quality constant* $\lambda \in [0, 1]$ that multiplies the satisfaction value of its operand by λ. We consider the logic LTL^∇, which extends LTL by the ∇_λ operator. The standard LTL operators are adjusted in LTL^∇ to values in $[0, 1]$: disjunctions are interpreted as $\max$, negation as subtraction from 1, and so on. For example, the satisfaction value of the formula $\psi_1 \vee \nabla_{\frac{1}{2}} \psi_2$ in a computation π is the maximum between the satisfaction value of ψ_1 in π, and $\frac{1}{2}$ the satisfaction value of ψ_2 in π. As a more elaborate example, consider a system that grants locks to a data structure. The system can grant either a read-only lock or a read-write lock. The quality of the system may be specified as $\mathsf{G}(\mathit{req} \rightarrow \mathsf{X}((\nabla_{\frac{3}{4}} \mathit{read\text{-}only}) \vee \mathit{read\text{-}write})$, implying that receiving a read-only lock satisfies the formula with value $\frac{3}{4}$, whereas receiving a read-write lock gives a higher satisfaction value of 1. In [2], the authors demonstrate the usefulness of the ability to specify quality and solve the model-checking and synthesis problems for LTL^∇.

Already in the Boolean setting, both model checking and synthesis rely on the specification to accurately reflect the designer's intention. One of the criticisms against formal method is that the latter challenge, of coming up with correct specifications, is not much easier than model checking or synthesis. Thus, formal methods merely shift the difficulty of developing correct implementations to that of developing correct specifications [13]. *Property assurance* is the activity of eliciting specifications that faithfully capture designer intent [8, 21]. One approach for property assurance is to challenge given specifications with sanity checks like non-validity, satisfiability, and vacuity [15]. More involved quality checks are studied in the PROSYD project [20] [1]. A second approach is that of automatic generation of specifications. This includes ideas from *learning*, where specifications given by means of automata are learned from a sample of behaviors tagged as good or bad [6, 17], methods based on a generation of specifications from basic *patterns* [11], and *specification mining* where specifications are generated by analyzing the runs of the given system [4]. In the novel quantitative setting, there is (yet) no experience in specification design nor tools or methods for property assurance. In this paper, we study the problem of automatically generating the quality layer in LTL^∇ formulas.

The design of an LTL^∇ formula typically starts with an LTL formula on top of which the designer adds the parameterized ∇ operators. The underlying assumption behind our approach is that in practice it is easier for a designer to provide desired satisfaction values in representative computations than to come up with quality constants that capture his intuition of high and low quality. This resembles the classical process of learning a hypothesis from tagged samples. Formally, an LTL^∇ *query* is an LTL^∇ formula in which some of the quality constants are replaced by variables. A *path constraint* is a pair $\langle \pi, I \rangle$ where π is a lasso-shaped path and $I \subseteq [0, 1]$ is a closed interval. Consider an LTL^∇ query φ over variables in $\mathcal{X}$. For an assignment $f : \mathcal{X} \rightarrow [0, 1]$ we use φ^f to denote the LTL^∇ formula obtained from φ by replacing each variable $x \in \mathcal{X}$ by $f(x)$. The LTL^∇ query problem is

[1] A related line of research is that of specification debugging [5], where, in the process of model checking, counterexamples are automatically clustered together in order to make the manual debugging of temporal properties easier.

to find, given an $\mathrm{LTL}^{\triangledown}$ query φ and a set $\mathcal{C}$ of path constraints, an assignment f to the variables in φ so that φ^f satisfies all the constraints (or returns that no such assignment exists): for all $\langle \pi, I \rangle \in C$, the satisfaction value of φ^f in π is in the interval I. Note that I may (but need not) be a single point. Note that beyond the restrictions on the quality constants in φ^f that follow from the constraints, restrictions may be induced also by repeated occurrences of the same variable. Subtle connections between different constants can be specified too, using nesting. In practice, however, most queries are simple (that is, each variable appear only once) and free of nesting.

As an example, consider the following specification: "After a request, an ack should ideally be given immediately and hold for two time units. An ack that holds only for one time unit is also acceptable, provided that it is given within two time units". A designer that wants to formalizes "ideally" and "acceptable" may have a clear idea that he wants to upper bound the satisfaction value of a policy with a single time unit ack by $\frac{3}{4}$ but may find it difficult to come up with the exact "penalty" for a delay in this case. This situation is captured by the following $\mathrm{LTL}^{\triangledown}$ query:

$$\mathsf{G}(req \rightarrow ((\mathsf{X}ack \wedge \mathsf{XX}ack) \vee \triangledown_{\frac{3}{4}}(\triangledown_x(\mathsf{X}ack) \vee \triangledown_y(\mathsf{XX}ack)))).$$

The satisfaction value of an induced $\mathrm{LTL}^{\triangledown}$ formula in a computation with an ideal ack policy is 1. In a computation with a single time unit ack it is at most $3/4$, to be further tuned down by the assignments to x and y. The designer does find it easy to grade given behaviours. For example, he may declare that a computation $(\{req\}, \{ack\})^\omega$ is not that bad, and satisfies the specification with quality $\frac{3}{4}$. Also, the path $(\{req\}, \emptyset, \{ack\})^\omega$ satisfies it with quality in $[\frac{1}{3}, \frac{1}{2}]$. A solution to the corresponding $\mathrm{LTL}^{\triangledown}$ query problem suggests an assignment to x and y that satisfies the designer's constraints. For example, $x = 1$ and $y = \frac{1}{2}$.

Before we continue to describe our results, let us review other settings with partially-specified systems or specifications. In the Boolean setting, reasoning about partially-specified systems is useful in automatic partial synthesis [22] and program repair [14]. From the other direction, partially-specified specifications are used for system exploration. In particular, in *query checking* [10], the specification contains variables, and the goal is to find an assignment to the variables with which the explored system satisfies the specification. While the formulation of the problem is similar, the motivation is very different, as the goal is to explore, synthesize, or reason about the system, whereas our goal here is automatic generation of specifications. The fact we consider the quantitative setting makes the underlying considerations and algorithms very different too. In the quantitative setting, related work includes parameterized weighted containment [7], where a partially specified weighted automaton is given, and the goal is to find an assignment to the missing weights such that a containment constraint is met. An orthogonal research direction is that of parametric real-time reasoning [3]. There, the quantitative nature of the automata origins from real-time constraints, the semantics is very different, and the goal is to find restrictions on the behavior of the clocks such that the automata satisfy certain properties.

We start by showing that in general, the $\mathrm{LTL}^{\triangledown}$ query problem is NP-hard. Checking whether a suggested assignment satisfies the set of constraints can be done in polynomial time, suggesting that the problem is in NP. One, however, also has to consider the domain and representation of the interval constraints. For example, the only solution to the query $\triangledown_x \triangledown_x p$ and the constraint $\langle \{p\}^\omega, \frac{1}{2} \rangle$ assigns to x the value $\sqrt{2}$, which is irrational and thus does not have a finite representation in a binary expansion. For common queries, in

particular ones in which each variable appears only once and without nesting of variables, we are able to prove that a "short" satisfying assignment exists, making the problem NP-complete. Hardness in NP holds already for the simple LTL^{∇} queries, and even for a single constraint.

We then proceed to study a fragment of the problem, where the LTL^{∇} queries are simple and the set of constraints includes a single computation. We show that in this case, the problem can be solved in polynomial time.[2] Our polynomial algorithm is based on an analysis of the search space, showing that the infinitely many possible assignments to each of the variables x in the query can be rounded up to linearly many ones, depending on the desired satisfaction value and the structure of the formula inside which x is nested. The induced space of possible assignments can be then searched efficiently.

Finally, we use the case of a single constraint in a heuristic for the general case. We suggest and explore three heuristics: yak yak yak To demonstrate the usefulness of our algorithms, we implemented our algorithms and tested them on examples (partially from [12]), upon which we added ∇_λ operators. We show that To do: bla bla...

2 The Logic LTL^{∇}

The logic LTL^{∇} is a multi-valued logic that extends the linear temporal logic LTL with a parameterized quality operator ∇_λ. The logic, along with model-checking and synthesis algorithms for it, was introduced in [2]. We start by defining its syntax and semantics. Let AP be a set of Boolean atomic propositions[3]. An LTL^{∇} formula is one of the following:

- $\mathtt{True}$, $\mathtt{False}$, or p, for $p \in AP$.
- $\neg\varphi$, $\varphi \vee \psi$, $\nabla_\lambda\varphi$, $\mathsf{X}\varphi$, or $\varphi\mathsf{U}\psi$, for LTL^{∇} formulas φ and ψ, and a *quality constant* $\lambda \in [0,1]$.

The semantics of LTL^{∇} is defined with respect to infinite computations over AP. Each position in the computation corresponds to a valuation to the atomic propositions, thus a computation is a word $\pi = \pi_0, \pi_1, \ldots \in (2^{AP})^\omega$. We use π^i to denote the suffix $\pi_i, \pi_{i+1}, \ldots$ of π. The semantics maps a computation π and an LTL^{∇} formula φ to the satisfaction value of φ in π, denoted $[\![\pi, \varphi]\!]$. The satisfaction value is in $[0,1]$, defined by induction on the structure of φ as described in Table 1 below. As with LTL, we use $\mathsf{F}\psi$ ("eventually") and $\mathsf{G}\psi$ ("always") as abbreviations for $\mathtt{True}\mathsf{U}\psi$ and $\neg\mathsf{F}\neg\psi$, respectively, as well as the standard Boolean abbreviations $\wedge$ and $\rightarrow$.

Evaluating LTL^{∇} *formulas on lasso computations* We say that a computation π is a *lasso* if $\pi = u \cdot v^\omega$, for finite computations $u, v \in (2^{AP})^*$ with $v \neq \epsilon$. We refer to u as the *prefix* of the lasso and to v as its *cycle*. The standard bottom-up labeling algorithm for model checking LTL formulas with respect to lasso computations can be easily extended to LTL^{∇}. The algorithm is based on the simple observation that if $[\![\pi^i, \psi]\!]$ is known for all $i \geq 0$ and subformulas ψ of φ, then it is possible to calculate, in time linear in $|u| + |v|$, the values $[\![\pi^i, \varphi]\!]$, for all $i \geq 0$. Indeed, the periodicity of π implies that there are only $|u| + |v|$

[2] We note that both requirements, of simple queries and a singleton constraint are needed; removing one of them we are back to NP-hardness.

[3] As discussed in Remark 1, it is possible to extend the definition as well as our results to weighted atomic propositions with values in $[0,1]$.

Formula	Satisfaction Value
$[\![\pi, \texttt{True}]\!]$	1
$[\![\pi, \texttt{False}]\!]$	0
$[\![\pi, p]\!]$	1 if $p \in \pi_0$ 0 if $p \notin \pi_0$
$[\![\pi, \neg\varphi]\!]$	$1 - [\![\pi, \varphi]\!]$
$[\![\pi, \varphi \vee \psi]\!]$	$\max([\![\pi, \varphi]\!], [\![\pi, \psi]\!])$
$[\![\pi, \nabla_\lambda \varphi]\!]$	$\lambda \cdot [\![\pi, \varphi]\!]$
$[\![\pi, \mathsf{X}\varphi]\!]$	$[\![\pi^1, \varphi]\!]$
$[\![\pi, \varphi \mathsf{U} \psi]\!]$	$\max_{i \geq 0}\{\min\{[\![\pi^i, \psi]\!], \min_{0 \leq j < i} [\![\pi^j, \varphi]\!]\}\}$

Table 1. The semantics of LTL^∇.

different suffixes to consider, and, by the semantics of LTL^∇, the satisfaction value of φ can be easily inferred from the satisfaction value of its subformulas. The only non-trivial case is when $\varphi = \psi_1 \mathsf{U} \psi_2$, but also there, one can start with the satisfaction value of ψ_2 and then repeatedly go back the lasso checking for every suffix whether, taking the satisfaction value of ψ_1 into an account, it is worthwhile to postpone the satisfaction of the eventuality. To conclude, we have the following.

Proposition 1. *Given an* LTL^∇ *formula* φ *and finite computations* $u, v \in (2^{AP})^*$*, calculating* $[\![u \cdot v^\omega, \varphi]\!]$ *can be done in time* $O(|\varphi| \cdot (|u| + |v|))$.

2.1 LTL^∇ queries

Let $\mathcal{X}$ be a finite set of variables. An LTL^∇ *query* (over $\mathcal{X}$) is an LTL^∇ formula in which some of the quality constants are replaced with variables from $\mathcal{X}$. For example, $\varphi = \mathsf{G}(req \rightarrow ((\nabla_{x_1} X(read \vee \nabla_{x_2} X\, read)) \vee (\nabla_{x_3} X\, write) \vee (\nabla_{\frac{3}{4}} halt)))$ is an LTL^∇ query over $\{x_1, x_2, x_3\}$. We say that an LTL^∇ query is *simple* if each of its variables occurs only once. The *depth* of an LTL^∇ query φ is the maximal nesting depth of variables in φ. For example, φ above is simple and is of depth 2. Note that, as in φ above, not all quality constraints are replaced by variables. For an LTL^∇ query φ, we denote by $var(\varphi)$ the set of variables $x \in \mathcal{X}$ such that $\nabla_x \psi$ is a subformula of φ. Given an assignment $f : \mathcal{X} \rightarrow [0, 1]$, we define φ^f to be the LTL^∇ formula obtained from φ by replacing every occurrence of $x \in \mathcal{X}$ with $f(x)$. Note that an assignment f as above prioritizes the different possible ways to satisfy the specification. In φ above, the assignment to x_1 and x_3 reflects the priority of the designer as to whether a read lock or a write lock is granted after a request, and the assignment to x_2 reflects the cost of a delayed read lock.

Consider an LTL^∇ query φ. A *path constraint* is a pair $\langle \pi, I \rangle$ such that $\pi \in (2^{AP})^\omega$ and $I \subseteq [0, 1]$ is a closed interval; that is, $[a, b]$ for $0 \leq a \leq b \leq 1$. A *lasso constraint* is a path constraint in which π is a lasso. When the interval I is a single point, thus $a = b$, we only state the point in the specification of the constraint.

The LTL^∇ query problem is to decide, given an LTL^∇ query φ and a set $\mathcal{C}$ of lasso constraints, whether there exists an assignment f to $var(\varphi)$ such that $[\![\pi, \varphi^f]\!] \in I$ for all $\langle \pi, I \rangle \in \mathcal{C}$. We then say that the assignment f is a *solution* to $\langle \varphi, \mathcal{C} \rangle$.

2.2 Generation of LTL$^\triangledown$ queries

Typically, LTL$^\triangledown$ formulas are obtained from LTL formulas by adding the $\triangledown_\lambda$ operator to various components. A common pattern for LTL specifications is a conjunction of specifications $\alpha \wedge \beta$. We observe that introducing the $\triangledown_\lambda$ naively may result in an undesirable behavior: consider the formula $\alpha \wedge \triangledown_{\frac{3}{4}} \beta$. The designer may have meant that β should have a lower effect on the satisfaction value, but according to the semantics, this just limits the satisfaction value to $\frac{3}{4}$ and makes the contribution of α, for values above $\frac{3}{4}$, irrelevant. We now demonstrate two sound methods for adding $\triangledown_\lambda$ operators.

The first method is based on the observation that $\triangledown_\lambda$ works best with disjunction, in the sense that if one of the components is under $\triangledown_\lambda$, it does not limit the satisfaction value for the entire formula. Thus, we can break the conjunction into a disjunction of cases, and apply the $\triangledown_\lambda$ operator. For example, $\alpha \wedge \beta$ becomes $\triangledown_{\lambda_1}(\alpha \wedge \beta) \vee (\triangledown_{\lambda_2} \alpha) \vee (\triangledown_{\lambda_3} \beta)$. After this transformation, the intuition that each component is graded differently is met.

The second method is to use negations to invert the behavior of $\triangledown_\lambda$. Consider the formula $\neg \triangledown_\lambda \neg \varphi$ for some formula φ. It holds that $[\![\pi, \neg \triangledown_\lambda \neg \varphi]\!] = 1 - \lambda(1 - [\![\pi, \varphi]\!])$. In particular, if $[\![\pi, \varphi]\!] = 1$, then this value is 1, and if $[\![\pi, \varphi]\!] = 0$, this value is $1 - \lambda$. For the formula $\alpha \wedge \beta$, if α is not critical, we can specify e.g. $(\neg \triangledown_\lambda \neg \alpha) \wedge \beta$. This captures the intuition that if α and β both hold, the satisfaction value is 1, but if α does not hold, the satisfaction value does not decrease to 0.

3 Solving the LTL$^\triangledown$ Query Problem

In this section we study the complexity of the LTL$^\triangledown$ query problem and show that it is NP-hard. As follows from Proposition 1, given an LTL$^\triangledown$ query φ, a set $\mathcal{C}$ of lasso constraints, and an assignment $f : var(\varphi) \to [0, 1]$, it is possible to check in linear time whether f is a solution for $\langle \varphi, \mathcal{C} \rangle$. Indeed, for each of the lasso constraints $\langle \pi, I \rangle \in \mathcal{C}$ we can calculate $[\![\pi, \varphi^f]\!]$ and verify that it is in I. This suggests that the LTL$^\triangledown$ query is in NP, as given a witness assignment f, we can verify it efficiently. Membership in NP, however, also requires the witness f to be polynomial in the φ and $\mathcal{C}$.

The latter requirement add to the picture considerations like the domain and presentation of the interval constraints. A natural suggestion is to assume that all intervals I are of the form $[a, b]$ for rational numbers $0 \leq a, b \leq 1$, given by their binary expansion. As we now demonstrate, things are involved already in this case. To see why, consider the query $\triangledown_x \triangledown_x p$ and the constraint $\langle \{p\}^\omega, \frac{1}{2} \rangle$. The single solution to the problem is f with $f(x) = \sqrt{2}$. But $\sqrt{2}$ is irrational, and therefore its binary expansion is infinite. Thus, while it is possible that the problem is in NP, describing a witness for an input requires a more sophisticated way of encoding solution, which is of debatable interest to the CAV community. As good news, in Section 4.1 we show that for typical instances of the problem, namely simple queries of depth 1, short witnesses exist, making the problem NP-complete for them. The proof requires results we develop in Section 4. Here, we describe the lower bound.

Theorem 1. *The* LTL$^\triangledown$ *query problem is NP-hard.*

Proof. We describe a reduction from 3-SAT. Let $\theta = (l_1^1 \vee l_2^1 \vee l_3^1) \wedge ... \wedge (l_1^k \vee l_2^k \vee l_3^k)$ be a 3-CNF formula. We construct an LTL$^\triangledown$ query φ and a set $\mathcal{C}$ of constraints such that θ is satisfiable iff there is a solution for $\langle \varphi, \mathcal{C} \rangle$. Let $X = \{x_1, ..., x_m\}$ be the set of variables

that appear in θ. We define $AP = \{p_1, n_1, ..., p_m, n_m\}$ and $\mathcal{X} = \{y_1, z_1, \ldots, y_m, z_m\}$. Intuitively, the proposition p_i (resp. n_i) stands for "the variable x_i appears positively (resp. negatively) in the clause", and we define the query and the constraints so that the variable y_i (resp. z_i) is assigned 1 when x_i is assigned `True` (resp. `False`).

We define $\varphi = \mathsf{G}(\triangledown_{y_1} p_1 \vee \triangledown_{z_1} n_1 \vee ... \vee \triangledown_{y_m} p_m \vee \triangledown_{z_m} n_m)$. We first have to ensure that in every solution to the query, at least one of the variables $\{y_i, z_i\}$ gets value 0, for all $1 \leq i \leq m$. This is done by the constraint $\langle \pi_i, 0 \rangle$, with $\pi_i = \{p_i\}\{n_i\}(AP)^\omega$. Note that in order for $[\![\pi_i, \varphi]\!]$ to be 0, it must be that either $[\![\{p_i\}, \triangledown_{y_i} p_i]\!] = 0$ or $[\![\{n_i\}, \triangledown_{z_i} n_i]\!] = 0$, implying that indeed at least one of the variables $\{y_i, z_i\}$ has value 0.

The family of m constraints above guarantees that a solution f to the query induces a truth assignment to X: the variable x_i is assigned `True` iff $f(y_i) = 1$. It is left to ensure that f induces a satisfying assignment. This is done by the constraint $\langle \pi, 1 \rangle$, where $\pi = \{s_1^1, s_2^1, s_3^1\} \cdots \{s_1^k, s_2^k, s_3^k\} \cdot (AP)^\omega$ is such the i-th position corresponds to the i-th clause and ensures that at least one of its literals gets value `True`. Accordingly, s_j^i is p_t if $l_j^i = x_t$ and is n_t if $l_j^i = \neg x_t$. Note that in order for $[\![\pi, \varphi]\!]$ to be 1, it must be that $[\![\{s_1^i, s_2^i, s_3^i\}, \triangledown_{y_t} p_t]\!] = 1$ or $[\![\{s_1^i, s_2^i, s_3^i\}, \triangledown_{z_t} z_t]\!] = 1$, for some t such that x_t appears in the i-th clause. If x_t appears positively in the clause, then one of the s_j^i's is p_t, and if x_t appears negatively, then one of them is n_t. Thus, in a solution f, one of the corresponding variables – that is, y_t in the first case and n_t in the second, is assigned 1.

For example, let $\theta = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_3)$. Then, $\varphi = \mathsf{G}(\triangledown_{y_1} p_1 \vee \triangledown_{z_1} n_1 \vee \triangledown_{y_2} p_2 \vee \triangledown_{z_2} n_2 \vee \triangledown_{y_3} p_3 \vee \triangledown_{z_3} n_3)$, and $\mathcal{C}$ contains the constraints $\langle \{p_1, n_2, p_3\}\{n_1, p_2, p_3\}(AP)^\omega, 1 \rangle$ and $\langle \{p_i\}\{n_i\}(AP)^\omega, 0 \rangle$, for $i \in \{1, 2, 3\}$. The formula θ is satisfiable. For example, the satisfying assignment in which x_1 and x_2 are `True` and x_3 is `False` induces a solution to $\langle \varphi, \mathcal{C} \rangle$ in which $(y_1, z_1) = (1, 0)$, $(y_2, z_2) = (1, 0)$, and $(y_3, z_3) = (0, 1)$. It is easy to see that the reduction is polynomial.

Theorem 1 motivates a study of special easy cases of the LTL$^\triangledown$ query problem. Since the reduction in the proof of Theorem 1 uses a query with multiple constraints, a natural candidate is the case of a single constraint. Lemma 1 below hints that this case is not easier.

Lemma 1. *Let φ be a simple* LTL$^\triangledown$ *query over a set $\mathcal{X}$ of variables and let $\mathcal{C}$ be a set of lasso constraints of the form $\langle \pi, 1 \rangle$ or $\langle \pi, 0 \rangle$. Then, there exists an* LTL$^\triangledown$ *query φ' over $\mathcal{X}$ and a lasso π such that for every assignment $f : \mathcal{X} \to [0, 1]$, we have that f is a solution to $(\varphi', \{\langle \pi, 0 \rangle\})$ iff f is a solution to $\langle \varphi, \mathcal{C} \rangle$. In addition, the length of the prefix of π is the length of the longest prefix of a lasso in $\mathcal{C}$, and the length of its cycle is the* lcm *(least common multiple) of the lengths of the cycles in the lassos in $\mathcal{C}$.*

Proof. Let AP be the set of atomic propositions in φ, and let $\mathcal{C} = \{\langle u_1 \cdot v_1^\omega, c_1 \rangle, \ldots, \langle u_k \cdot v_k^\omega, c_k \rangle\}$. We define AP' as k disjoint copies of AP, thus $AP' = AP \times \{1, \ldots, k\}$. We define φ_j to be the LTL$^\triangledown$ query obtained from φ by replacing each atomic proposition $p \in AP$ by the atomic proposition $\langle p, j \rangle \in AP'$. Let $u, v \in 2^{AP'}$ be such that for all $1 \leq j \leq k$, the projection of $u \cdot v^\omega$ on $AP \times \{j\}$ agrees with $u_j \cdot v_j^\omega$. It is easy to define u and v as above by taking u of length $m = \max_i |u_i|$ and v of length $\ell = lcm(|v_1|, ..., |v_k|)$. Indeed, the labeling of u by $AP \times \{j\}$ is obtained by concatenating to u_j a prefix of v_j^ω of length $m - |u_j|$ and the labeling of v by $AP \times \{j\}$ is then obtained from v_j' by the corresponding shift of $v_j^{\ell/|v_j|}$.

Now, we define $\varphi' = \bigwedge_{j=1}^k \psi_j$, where ψ_j is either φ_j, in case $c_j = 1$, or is $\neg\varphi_j$, in case $c_j = 0$. Consider an assignment $f : \mathcal{X} \to [0, 1]$. Since φ' is defined as a conjunction, then

the constraint $\langle u \cdot v^\omega, 1\rangle$ is met for φ' iff $[\![u \cdot v^\omega, \psi_j^f]\!] = 1$ for all $1 \leq j \leq k$, which, by the definition of ψ_j, holds iff f is a solution to $\langle \varphi, \{\langle u_j \cdot v_j^\omega, c_j\rangle\}\rangle$. To conclude, f is a solution to $\langle \varphi', \{\langle u \cdot v^\omega, 1\rangle\}\rangle$ iff f is a solution to $\langle \varphi, \mathcal{C}\rangle$.

While the single lasso constructed in Lemma 1 may be exponential in the original constraints, examining the lassos that are used in the reduction in the proof of Theorem 1, we see that they all have cycles of length 1. Therefore, Lemma 1 together with the reduction there imply that the special case of a single constraint is not easy. Formally, we have the following.

Theorem 2. *The* $\mathrm{LTL}^\triangledown$ *query problem is NP-hard even for the case of a single lasso constraint.*

4 A Feasible Special Case

While Theorem 2 implies that the $\mathrm{LTL}^\triangledown$ query problem is hard already for a single constraint, the transformation described in the proof of Lemma 1 generates formulas that are not simple. Indeed, the transformation is based on a relation between multiple constraints and multiple occurrences of a variable. In this section we show that in a setting with both limitations, the $\mathrm{LTL}^\triangledown$ query problem can be solved efficiently. Formally, we prove the following.

Theorem 3. *The* $\mathrm{LTL}^\triangledown$ *query problem for simple queries and a single constraint can be solved in polynomial time.*

In Section 5, we show that Theorem 3 and the algorithm developed for its proof are useful in approximation and heuristic algorithms for the general case.

Let φ be a simple $\mathrm{LTL}^\triangledown$ query over AP and $\mathcal{X}$, and let π be a computation. Let $k = |\mathcal{X}|$. Consider the function $\mu_{\pi,\varphi} : [0,1]^k \to [0,1]$ defined by $\mu_{\pi,\varphi}(f) = [\![\pi, \varphi^f]\!]$.

We start with some useful observations.

Lemma 2. *For all computations* π *and* $\mathrm{LTL}^\triangledown$ *queries* φ*, the function* $\mu_{\pi,\varphi}$ *is continuous. That is, for every infinite sequence* $(a_n)_{n=1}^\infty$ *of points in* $[0,1]^k$ *such that* $\lim_{n\to\infty} a_n = a$*, it holds that* $\lim_{n\to\infty}(\mu_{\pi,\varphi}(a_n)) = \mu_{\pi,\varphi}(a)$.

Proof. By the semantics of $\mathrm{LTL}^\triangledown$, the satisfaction value of an $\mathrm{LTL}^\triangledown$ formula in a computation is obtained from the values 0 and 1 by repeated applications of the operations $\min, \max$, subtraction from 1, and multiplication by $\lambda \in [0,1]$. Since all these operations are continuous, so is the function $\mu_{\pi,\varphi}$.

Consider a variable $x \in \mathcal{X}$. Since φ is simple, the variable x is either *positive* in φ, in case the subformula $\triangledown_x \psi$ is in the scope of an even number of negation, or is *negative* in φ, otherwise. We refer to the positivity or negativity of x in φ as its *polarity* in φ.

Lemma 3. *For all computations* π *and* $\mathrm{LTL}^\triangledown$ *queries* φ*, the function* $\mu_{\pi,\varphi}$ *is monotonic in each variable. Specifically, for every variable* $x \in \mathcal{X}$*, if* x *is positive in* φ *then* $\mu_{\pi,\varphi}$ *is increasing with* x *and if* x *is negative in* φ *then* $\mu_{\pi,\varphi}$ *is decreasing with* x*.*

Proof. Among the four operations mentioned in the proof of Lemma 2, the operations $\min, \max$, and multiplication by $\lambda \in [0,1]$ are increasing with their operands, whereas subtraction from 1 is decreasing with its operand. Since the latter is applied whenever a negation is evaluated, the claim follows form the definition of the polarity of x.

We note that full proofs of Lemmas 2 and 3 involve an (easy) induction on the structure of φ.

The idea behind our polynomial algorithm is to limit the search space for a satisfying assignment. Before defining the limited search space, let us first observe that an $\mathrm{LTL}^{\triangledown}$ formula has finitely (in fact, linearly many) possible satisfaction values. We define the set of possible values of φ, denoted $val(\varphi)$, by induction on the structure of φ as follows.

- If $\varphi = p \in AP$, then $val(p) = \{0, 1\}$.
- If $\varphi = \psi_1 \vee \psi_2$ or $\varphi = \psi_1 U \psi_2$, then $val(\varphi) = val(\psi_1) \cup val(\psi_2)$.
- If $\varphi = \neg\psi$, then $val(\varphi) = \{1 - v : v \in val(\psi)\}$.
- If $\varphi = \mathsf{X}\psi$, then $val(\varphi) = val(\psi)$.
- If $\varphi = \triangledown_\lambda \psi$, then $val(\varphi) = \{\lambda \cdot v : v \in val(\psi)\}$.

It is easy to prove that for every path π it holds that $[\![\pi, \varphi]\!] \in val(\varphi)$.

We start by defining the limited search space for $\mathrm{LTL}^{\triangledown}$ queries of depth 1. We will later generalize the definition to all depths. Let φ be a simple $\mathrm{LTL}^{\triangledown}$ query of depth 1. For $x \in var(\varphi)$ and $c \in [0,1]$ we define the set of relevant values for x with respect to φ and c, denoted $val(x, \varphi, c)$, by induction on the structure of φ as follows.

- If $\varphi = \triangledown_x \psi$, for a $\mathrm{LTL}^{\triangledown}$ formula ψ, then $val(x, \varphi, c) = \{\frac{c}{v} : v \in val(\psi) \text{ and } v \geq c\}$. Note that since φ is of nesting depth 1, then ψ has no variables, and this is the base case for the induction.
- If $\varphi = \psi_1 \vee \psi_2$ or $\varphi = \psi_1 U \psi_2$, then $val(x, \varphi, c) = val(x, \psi_i, c)$, for the single $i \in \{1, 2\}$ such that $x \in var(\psi_i)$.
- If $\varphi = \mathsf{X}\psi$, then $val(x, \varphi, c) = val(x, \psi, c)$.
- If $\varphi = \neg\psi$, then $val(x, \varphi, c) = val(x, \psi, 1 - c)$.
- If $\varphi = \triangledown_\lambda \psi$, we distinguish between two cases. If $\lambda \geq c$, then $val(x, \varphi, c) = val(x, \psi, \frac{c}{\lambda})$. Otherwise, $val(x, \varphi, c) = \emptyset$.

Lemma 4 below justifies the restricted search space. Consider a value $u \in [0,1]$. Let $up_{x,\varphi,c}(u)$ and $down_{x,\varphi,c}(u)$ be the "rounding" up and down of u to the nearest value in $val(x, \varphi, c)$. Formally, $up_{x,\varphi,c}(u) = \min\{v : v \in val(x, \varphi, c) \text{ and } v \geq u\}$ and $down_{x,\varphi,c}(u) = \max\{v : v \in val(x, \varphi, c) \text{ and } v \leq u\}$.

Consider an assignment $f : \mathcal{X} \to [0,1]$. For a variable $x \in \mathcal{X}$, define the assignments $f^+_{x,\varphi,c}$ and $f^-_{x,\varphi,c}$ as the assignments obtained from f by leaving the assignments to all variables except x unchanged and rounding the value of x up or down to the nearest value in $val(x, \varphi, c)$. The decision whether to round the value of x up or down depends on the $+$ and $-$ indication as well as in the polarity of x in φ. Formally, we have the following.

$$f^+_{x,\varphi,c}(x) = \begin{cases} up_{x,\varphi,c}(f(x)) & \text{if } x \text{ is positive in } \varphi, \\ down_{x,\varphi,c}(f(x)) & \text{if } x \text{ is negative in } \varphi, \end{cases} \text{ and } f^-_{x,\varphi,c}(x) = \begin{cases} up_{x,\varphi,c}(f(x)) & \text{if } x \text{ is negative in } \varphi, \\ down_{x,\varphi,c}(f(x)) & \text{if } x \text{ is positive in } \varphi. \end{cases}$$

Lemma 4. *Consider a simple* $\mathrm{LTL}^{\triangledown}$ *query* φ *of depth* 1 *and an assignment* f *to* $var(\varphi)$*. Let* $c \in [0,1]$*, and let* π *be a computation. Then,*

1. *If* $[\![\pi, \varphi^f]\!] \leq c$*, then* $[\![\pi, \varphi^{f^+_{x,\varphi,c}}]\!] \leq c$.
2. *If* $[\![\pi, \varphi^f]\!] \geq c$*, then* $[\![\pi, \varphi^{f^-_{x,\varphi,c}}]\!] \geq c$.

Before we prove the lemma, let us note that by the monotonicity of $\mathrm{LTL}^{\triangledown}$ queries, increasing the value of a variable x that appears positively in φ can only increase the satisfaction value of φ (and dually for reducing the value of x or for the case of a variable that appears negatively). The claim in Lemma 4, however, is different and is much stronger, as it states that we can actually increase the value of a variable that appears positively without increasing the value of φ. More precisely, if the satisfaction value of φ^f in π is below c, then we can round the value of x up to the closest value in $val(x, \varphi, c)$ and still keep the satisfaction value below c.

Proof. The proof proceeds by induction on the structure of φ. We prove the first claim, the second is dual.

- If $\varphi = p \in AP$, then since there are no variables in φ, changing f does not affect its satisfaction value, and we are done.
- If $\varphi = \psi_1 \vee \psi_2$, assume w.l.o.g that $x \in var(\psi_1)$. Since $[\![\pi, \varphi^f]\!] \leq c$, then $[\![\pi, \psi_1^f]\!] \leq c$ and $[\![\pi, \psi_2^f]\!] \leq c$. Changing f to $f^x_{x,\varphi,c}$ does not affect the satisfaction value of ψ_2, and by the induction hypothesis, $[\![\pi, \psi_1^{f^+_{x,\psi_1,c}}]\!] \leq c$, so $[\![\pi, \varphi^{f^x_{x,\varphi,c}}]\!] \leq c$.
- If $\varphi = \mathsf{X}\psi$, then the claim follows easily from the induction hypothesis.
- If $\varphi = \neg\psi$, then $[\![\pi, \varphi^f]\!] \leq c$ implies $[\![\pi, \psi^f]\!] \geq 1 - c$. Observe that $f^+_{x,\varphi,c}$ agrees with $f^-_{x,\psi,1-c}$. Thus, by the induction hypothesis, applied to the fact $[\![\pi, \psi^f]\!] \geq 1-c$, we get that $[\![\pi, \psi^{f^-_{x,\psi,1-c}}]\!] \geq 1 - c$, or equivalently $[\![\pi, \psi^{f^+_{x,\varphi,c}}]\!] \geq 1 - c$. So, $[\![\pi, \varphi^{f^+_{x,\varphi,c}}]\!] \leq c$, and we are done.
- If $\varphi = \triangledown_\lambda \psi$, then $[\![\pi, \varphi^f]\!] \leq c$ implies $[\![\pi, \psi^f]\!] \leq c/\lambda$ (if $\lambda = 0$, then this is a degenerate case). By the definition of $val(x, \varphi, c)$, we have that $f^+_{x,\varphi,c}$ agrees with $f^+_{x,\psi,c/\lambda}$. Thus, by the induction hypothesis, we have that $[\![\pi, \psi^{f^+_{x,\varphi,c}}]\!] \leq c/\lambda$, so $[\![\pi, \varphi^{f^+_{x,\varphi,c}}]\!] \leq c$, and we are done.
- If $\varphi = \triangledown_x \psi$, then since φ is of depth 1, we know that ψ has no variables. Assume that $[\![\pi, \varphi^f]\!] \leq c$, and assume, by way of contradiction, that $[\![\pi, \varphi^{f^+_{x,\varphi,c}}]\!] > c$. Since $\mu_{\pi,\varphi}$ is continuous (Lemma 2), then by the intermediate value theorem there exists an assignment h that agrees with f and $f^+_{x,\varphi,c}$ on all variables but x and $[\![\pi, \varphi^h]\!] = c$. We claim that $h(x) \in val(x, \varphi, c)$. Indeed, since $[\![\pi, \varphi^h]\!] = c$, then $y \cdot [\![\pi, \psi]\!] = c$, so $h(x)/c \in val(\psi)$, which by definition implies that $h(x) \in val(x, \varphi, c)$. This, however, contradicts the definition of $f^+_{x,\varphi,c}(x)$ as the rounding of $f(x)$ to the closest value in $val(x, \varphi, c)$.
- If $\varphi = \psi_1 \mathsf{U} \psi_2$, we proceed as follows. By the semantics of U, the fact $[\![\pi, \varphi^f]\!] \leq c$ implies that for every $i \geq 0$, it holds that $\min\{[\![\pi^i, \psi_2^f]\!], \min_{0 \leq j < i}\{[\![\pi^j, \psi_1^f]\!]\}\} \leq c$. Thus, for every $i \geq 0$ either $[\![\pi^i, \psi_2^f]\!] \leq c$ or there exists $0 \leq j < i$ such that $[\![\pi^j, \psi_1^f]\!] \leq c$. Assume $x \in var(\psi_1)$ and consider the assignment $f^+_{x,\varphi,c}$. For every $i \geq 0$, if $[\![\pi^i, \psi_2^f]\!] \leq c$ then $[\![\pi^i, \psi_2^{f^+_{x,\varphi,c}}]\!] \leq c$ (since the value is unchanged). If $[\![\pi^j, \psi_1^f]\!] \leq c$ for some $j < i$, then by the induction hypothesis, we have that $[\![\pi^j, \psi_1^{f^+_{x,\varphi,c}}]\!] \leq c$. We conclude that for every $i \geq 0$, we have that $\min\{[\![\pi^i, \psi_2^{f^+_{x,\varphi,c}}]\!], \min_{0 \leq j < i}\{[\![\pi^j, \psi_1^{f^+_{x,\varphi,c}}]\!]\}\} \leq c$, implying that $[\![\pi, \varphi^{f^+_{x,\varphi,c}}]\!] \leq c$. The case where $x \in var(\psi_2)$ is similar.

We can now prove that the restriction of the search space to values in $val(x, \varphi, c)$ is allowed.

Lemma 5. *Let φ be a simple $\mathrm{LTL}^{\triangledown}$ query of depth 1, let $c \in [0,1]$, and let π be a path. If there exists an assignment f such that $[\![\pi, \varphi^f]\!] = c$, then there also exists an assignment g such that for every $x \in var(\varphi)$ it holds that $g(x) \in val(x, \varphi, c)$ and $[\![\pi, \varphi^g]\!] = c$.*

Proof. Consider a simple $\mathrm{LTL}^{\triangledown}$ query φ of depth 1 and an assignment f to $var(\varphi)$. Let $c \in [0,1]$, and let π be a computation. By the monotonicity of $\mu_{\pi,\varphi}$, Lemma 4 implies that if $[\![\pi, \varphi^f]\!] = c$, then $[\![\pi, \varphi^{f^+_{x,\varphi,c}}]\!] = c$.

Let f be such that $[\![\pi, \varphi^f]\!] = c$. The assignment g is obtained by repeating the following process all variables $x \in var(\varphi)$ in an arbitrary order: if $f(x) \in val(x, \varphi, c)$, then $g(x) = f(x)$. Otherwise, we define $g(x)$ to be $f^+_{x,\varphi,c}(x)$. By the above, $[\![\pi, \varphi^g]\!] = c$.

Consider a simple $\mathrm{LTL}^{\triangledown}$ query φ over $\mathcal{X}$. By Lemma 5, the search for a solution f to a single constraint with a point interval c involves a search in finitely many possible assignments – these that map each variable $x \in var(\varphi)$ to values in $val(x, \varphi, c)$. By Lamma 3 (monotonicity of assignments), the search can combine a binary search for the assignment for each $x \in var(\varphi)$ with an ordering of the different variables. We will get back to this point when we describe our experimental results in Section 6.

The search described above assumes simple queries of depth 1 and constraints with point intervals. We now remove both assumptions. Let f^{max} and f^{min} be the assignments that maximizes and minimizes the value of φ. Thus, $f^{max}(x)$ is 1 if x appears positively in φ and is 0 otherwise, and dually for f^{min}. We define the $\mathrm{LTL}^{\triangledown}$ formulas $\varphi^{max} = \varphi^{f^{max}}$ and $\varphi^{min}\varphi^{f^{min}}$. Note that $\triangledown_1\psi$ and $\triangledown_0\psi$ subformulas can be replaced by ψ and `False`, respectively.

For a simple $\mathrm{LTL}^{\triangledown}$ query φ (of an arbitrary depth), let $\varphi^\star$ be the $\mathrm{LTL}^{\triangledown}$ formula obtained from φ by replacing every subformula of the form $\triangledown_x\psi$ by $\triangledown_x\psi^{max}$. Note that $\varphi^\star$ is of depth 1. By Lamma 3 (monotonicity of assignments), for every computation π, $\mathrm{LTL}^{\triangledown}$ query ψ, and assignment f, we have $[\![\pi, \psi^f]\!] \leq [\![\pi, \psi^{max}]\!]$. Thus, replacing ψ by ψ^{max} may cause the satisfaction value of ψ to go above a desired bound. As we show below, however, in this case we can play with the assignment to x in order "tune down" the satisfaction value of ψ^{max}.

Lemma 6. *Let φ be a simple $\mathrm{LTL}^{\triangledown}$ query, and let $\langle \pi, I \rangle$ be a constraint. The $\mathrm{LTL}^{\triangledown}$ query $\langle \varphi, \langle \pi, I \rangle \rangle$ has a solution iff the query $\langle \varphi^\star, \langle \pi, I \rangle \rangle$ has a solution.*

Proof. First, since $\varphi^\star$ is obtained by a partial assignment to the variables of φ, then if $(\varphi^\star, \langle \pi, I \rangle)$ has a solution, so does $(\varphi, \langle \pi, I \rangle)$. For the other direction, let f be an assignment such that $[\![\pi, \varphi^f]\!] \in I$. Consider the satisfaction value of each subformula $\triangledown_x\psi$ of φ. By the semantics of $\mathrm{LTL}^{\triangledown}$, we have that $[\![\pi, (\triangledown_x\psi)^f]\!] = f(x) \cdot [\![\pi, \psi^f]\!]$. By Lemma 3, we have that $[\![\pi, \psi^f]\!] \leq [\![\pi, \psi^{max}]\!]$. If the latter is not an equality, consider an assignment g that is identical to f except that the variables in $var(\psi)$ are as in ψ^{max} and $g(x) = \frac{[\![\pi,\psi^f]\!]}{[\![\pi,\psi^{max}]\!]}$ (note that $[\![\pi, \psi^f]\!] < [\![\pi, \psi^{max}]\!]$ implies that $[\![\pi, \psi^{max}]\!] \neq 0$). We now have $[\![\pi, (\triangledown_x\psi)^g]\!] = [\![\pi, (\triangledown_x\psi)^f]\!]$. Thus, $[\![\pi, (\varphi^\star)^g]\!] = [\![\pi, \varphi^f]\!] \in I$, so $\langle \varphi^\star, \langle \pi, I \rangle \rangle$ has a solution.

Lemma 6 implies that we can use our algorithm also for formulas of depth greater than 1 by applying the algorithm to $\varphi^\star$. It is left to extend the algorithm to handle interval

constraints. That is, given a simple LTL$^\triangledown$ query φ and a lasso constraint $\langle \pi, I \rangle$ such that $I \subseteq [0,1]$, our goal is to decide whether $\langle \varphi, \langle \pi, I \rangle \rangle$ has a solution. We do this as follows. First, compute $a = [\![\pi, \varphi^{min}]\!]$ and $b = [\![\pi, \varphi^{max}]\!]$. If $I \cap [a,b] = \emptyset$, then, as $[\![\pi, \psi^{min}]\!] \leq [\![\pi, \psi^f]\!] \leq [\![\pi, \psi^{max}]\!]$, we can conclude that there is no solution to $\langle \varphi, \langle \pi, I \rangle \rangle$. Otherwise, by the continuity of $\mu_{\pi,\varphi}$, every value in $c \in [a,b]$ can be attained by $\mu_{\pi,\varphi}$, so one can choose any value $c \in I \cap [a,b]$, and find a solution to $\langle \varphi, \langle \pi, c \rangle \rangle$.

Remark 1. Our definition of computations assumes Boolean atomic propositions. It is easy to extend our setting and results to computations over *weighted atomic propositions*. Let WP be a finite set of weighted atomic propositions over some domain D. The domain D may be infinite (say, the natural numbers) and different propositions may be over different domains. Each position in the computation is then a function in D^{WP}, and the semantics of LTL$^\triangledown$ is as in the Boolean case, except that $[\![\pi, p]\!]$, for $p \in WP$ is $\pi_0(p)$. The solution of the corresponding LTL$^\triangledown$ query is similar to the one described above, except that the definition of $val(\psi)$, and consequently also $val(x, \varphi, c)$, should be adjusted to reflect the fact the weighted propositions in ψ can take values in D. Since each lasso commutation has only finitely many positions, the number of the values to be considered is still linear in the input to the problem.

4.1 NP completeness of the LTL$^\triangledown$ query problem for simple queries

Lemma 5 also gives us an upper bound to complete Theorem 1 for simple LTL$^\triangledown$ queries of depth 1. Given an LTL$^\triangledown$ query φ, a set of constraints $\mathcal{C} = \langle \langle \pi_i, [a_i, b_i] \rangle \rangle_{i=1}^m$, and a variable x, we define $val(x, \varphi, \mathcal{C}) = \bigcup_{i=1}^m val(x, \varphi, b_i)$. That is, $val(x, \varphi, \mathcal{C})$ includes the restricted search space of the upper end points of the intervals in the constraints in $\mathcal{C}$.

Lemma 7. *Let φ be a simple* LTL$^\triangledown$ *query of depth* 1*, and let $\mathcal{C}$ be a set of constraints. If there exists a solution f for $\langle \varphi, \mathcal{C} \rangle$, then there also exists a solution g such that for every $x \in var(\varphi)$, it holds that $g(x) \in val(x, \varphi, \mathcal{C})$.*

Proof. Let $\mathcal{C} = \{\langle \pi_i, [a_i, b_i] \rangle\}_{i=1}^m$ and let f be a solution for $\langle \varphi, \mathcal{C} \rangle$. Let $D = \{b_1, ..., b_m\}$, and consider a variable x. If $f(x) \notin val(x, \varphi, \mathcal{C})$, then in particular $f(x) \notin val(x, \varphi, d)$ for all $d \in D$. Let $\widehat{d} = \arg\min \{|f^+_{x,\varphi,d}(x) - f(x)| : d \in D\}$. By definition, $f^+_{x,\varphi,\widehat{d}}(x) \in val(x, \varphi, \mathcal{C})$. We claim that the assignment $g = f^+_{x,\varphi,\widehat{d}}$ is also a solution. Indeed, by Lemma 4, for every constraint $\langle \pi_i, [a_i, b_i] \rangle$ it holds that $a_i \leq [\![\pi_i, \varphi^f]\!] \leq b_i$, so $a_i \leq [\![\pi_i, \varphi^{g_i}]\!] \leq b_i$, where $g_i = f^+_{x,\varphi,b_i}$. For every $1 \leq i \leq m$, the assignments g_i and f, when regarded as points in $[0,1]^{|\mathcal{X}|}$, define a line segment (that is, $\{\alpha f + (1-\alpha) g_i : \alpha \in [0,1]\}$). Since g_i and f differ only in their assignment to x, then so do all the points along the segment. Thus, all the segments have some overlapping. The assignment g, by our choice of $\widehat{d}$, is in the overlap defined by f and g_i for every $1 \leq i \leq m$. Thus, by Lemma 3 (monotonicity of μ), it holds that $a_i \leq [\![\pi_i, \varphi^g]\!] \leq b_i$, so g is a solution. By repeating this process to each variable, we conclude the proof.

Since the binary expansion of the values in $val(x, \varphi, \mathcal{C})$ is polynomial in φ and $\mathcal{C}$ (with intervals given by their binary expansion), Lemma 7 implies that a witness solution to a simple LTL$^\triangledown$ query of depth 1 is polynomial. Since witnesses can be verified in linear time, we can conclude with the following.

Theorem 4. *The* LTL$^\triangledown$ *query problem for simple queries of depth* 1 *is NP-complete.*

5 Approximations and Heuristics

In this section we discuss two approximation schemes for the $\mathrm{LTL}^{\triangledown}$ query problem. The motivation is twofold. First, our approximating algorithms run in polynomial time, whereas, as studied in Section 3, the problem is NP-hard. Second, in case a query does not have a solution, it is helpful to find an assignment that approximates the solution. In order to study approximations, we should first formalize the $\mathrm{LTL}^{\triangledown}$ query problem as an optimization problem. This can be done in several ways. We consider here two common approaches. Consider an $\mathrm{LTL}^{\triangledown}$ query φ and a set $\mathcal{C}$ of constraints.

- In the *constraints-count* approach we seek an assignment that maximizes the number of constraints that are satisfied. Formally, for an assignment $f : Var(\varphi) \to [0,1]$, let $count(f) = |\{\langle \pi, I\rangle \in \mathcal{C} : [\![\pi, \varphi^f]\!] \in I\}|$. Then, we seek f for which $count(f)$ is maximal. Clearly, $count(f) = |\mathcal{C}|$ iff f is a solution.
- In the *distance-minimization* approach we seek an assignment that minimizes the distances from the intervals in $\mathcal{C}$. Formally, for an assignment $f : Var(\varphi) \to [0,1]$ and a constraint $\langle \pi, I\rangle$, let $\mathrm{loss}(f, \langle \pi, I\rangle) = \min\{([\![\pi, \varphi^f]\!] - x)^2 : x \in I\}$. That is, $\mathrm{loss}(f, \langle \pi, I\rangle)$ is the distance in norm L_2 from $[\![\pi, \varphi^f]\!]$ to I. Then, we seek f that minimizes $\sum_{\langle \pi, I\rangle \in \mathcal{C}} \mathrm{loss}(f, \langle \pi, I\rangle)$. Clearly, the sum of losses is 0 iff f is a solution.

Since, in both approaches, an assignment that is a solution can be characterized by means of an optimal approximation, the problem of finding an optimal approximation is NP-hard. As Theorem 2 shows, the $\mathrm{LTL}^{\triangledown}$ query problem is NP-hard even when a single constraint is allowed. Accordingly, since a nontrivial approximation of the the number of satisfied constraints must be greater than 0, it is NP-hard to even approximate the problem (to any ratio) under the constraint-count approach.

Our use of $\mathrm{LTL}^{\triangledown}$ queries for generating quality specifications makes the constraint-count approach less appealing. Indeed, the constraint-count approach takes us back to the Boolean setting. We still describe here this approach, as it easily suggests a naive approximation for the case the constraints are all pure upper- or lower-bounds:

Theorem 5. *For simple* $\mathrm{LTL}^{\triangledown}$ *queries and constraints whose intervals are of the form* $[0, b]$ *or* $[a, 1]$*, the constraint-count optimization problem has a* $\frac{1}{2}$*-approximation polynomial-time algorithm.*

Proof. Consider a simple $\mathrm{LTL}^{\triangledown}$ query φ, and let $\langle \pi, I\rangle$ be a constraint. By Lemma 3, if $I = [0, b]$, for $b \in [0, 1]$, then if there exists an assignment f such that $[\![\pi, \varphi^f]\!] \in I$, then $[\![\pi, \varphi^{min}]\!] \in I$. Similarly, if $I = [a, 1]$, for $a \in [0, 1]$, then $[\![\pi, \varphi^{max}]\!] \in I$. Now, given a set $\mathcal{C}$ of constraints of the above forms, we check which constraints are satisfied with f^{max} and which are satisfied with f^{min}. By our observation above, the optimal solution satisfies at most the union of the two, and thus one of f^{max} or f^{min} (the one for which the count is higher) satisfies at least $\frac{1}{2}$ of the maximal number of satisfiable constraints.

We suggest a polynomial-time heuristic algorithm, based on our ability to solve the $\mathrm{LTL}^{\triangledown}$ query problem efficiently for a single constraint (Theorem 3). Given a simple $\mathrm{LTL}^{\triangledown}$ query φ and a set $\mathcal{C}$ of constraints, we find, for every constraint $\langle \pi, I\rangle \in \mathcal{C}$, an assignment $f_{\langle \pi, I\rangle}$ such that $[\![\pi, \varphi^{f_{\langle \pi, I\rangle}}]\!] \in I$. Let $F = \{f_{\langle \pi, I\rangle} : \langle \pi, I\rangle \in \mathcal{C}\}$. There are several ways to obtain a single assignment from F. Our algorithm uses the following three.

– *Minimum assignment*, denoted f_{min}: For every $x \in var(\varphi)$, if x is positive in φ, then $f_{min}(x) = \min_{f \in F} f(x)$; otherwise $f_{min}(x) = \max_{f \in F} f(x)$.
– *Maximum assignment*, denoted f_{max}: For every $x \in var(\varphi)$, if x is positive in φ, then $f_{max}(x) = \max_{f \in F} f(x)$; otherwise $f_{max}(x) = \min_{f \in F} f(x)$.
– *Center of gravity*, denoted f_{CoG}: For every $x \in var(\varphi)$, we define $f_{CoG}(x) = \frac{1}{|F|} \sum_{f \in F} f(x)$. It is easy to prove that the center of gravity is the point that minimizes the L_2 distance from $f_1, ..., f_m$. That is, $f_{CoG} = \arg\min \{\sum_{i=1}^{m} ||f - f_i||^2\}$ (where $|| \cdot ||_2$ is the L_2 norm).

Yak yak 1: Add the example on how the three approaches may not be good.

6 Experimental Results

In this section we present experimental results demonstrating our heuristic algorithm for solving the LTL$^{\triangledown}$ query problem. We evaluate the quality of the heuristic in the two optimization approaches, namely constraint-counting and distance-minimization.

As our benchmark we use LTL formulas from [12], to which we add a quality layer, as well as queries constructed manually. As constraints, we use fair paths in the automaton for the corresponding LTL formulas as well as manually generated computations. An example of a query we took from [12] is a specification for a traffic light. The atomic propositions are $\{N, E, W, S\}$, standing for a green light for traffic coming from North, East, West, and South, respectively. We assume traffic crosses the junction and make no turns, and so the specification allows N and S, as well as W and E, to hold simultaneously, but not N and W, nor S and E, and so on. Thus, the specification includes the property $\psi = (G(N \vee S) \rightarrow (\neg E \wedge \neg W) \wedge G(E \vee W) \rightarrow (\neg N \wedge \neg S))$. We want the traffic light to direct the traffic efficiently, thus we require that at least one direction to has a green light. This is specified by the conjunction $G\theta$, for $\theta = (S \vee N \vee W \vee E)$. We may also be satisfied by $FG\theta$. But while ψ is a crucial safety requirement, the conjunctions with θ only concerns the efficiency of the traffic light. Thus, we can tune them done using the LTL$^{\triangledown}$ formula $\psi \wedge \blacktriangledown_{0.4} G\theta \wedge \blacktriangledown_{0.9} FG\theta$. Note that we prefer a junction that is never empty over a junction that is only eventually never empty. But this is not the end of the story. We may want to prioritize the different directions. Deciding the priorities and their combination with the external tuning down of the "efficiency requirement" may be a difficult task. So, we replace θ by $\theta' = (\blacktriangledown_{x_1} S) \vee (\blacktriangledown_{x_2} N) \vee (\blacktriangledown_{x_3} W) \vee (\blacktriangledown_{x_3} E)$, leaving the priorities as variables. The obtained LTL$^{\triangledown}$ query is φ_4 in the table. Examples of constraints we use are $\langle(\{N, S\}, \{E, W\})^{\omega}, 1\rangle$ and $\langle(\{N, S\}, \emptyset, \{E, W\})^{\omega}, 0.4\rangle$.

We have implemented the algorithm in Python and ran it on an Intel® Core i5 2.53GHz machine. The code can be found in: http://www.cs.huji.ac.il/∼guya03/CAV13/.

Since the designer needs to reason on the paths he supplies to our tool, the input constraints are typically short. However, in order to make sure our tool can handle long paths, we ran a random generated path of length 120 on a query with 8 parameters. It took 11 seconds to perform a search for a possible assignment, and reach the (un-surprising) result that there is no possible assignment. So we believe that our tool can handle constraints of any practical length.

In Table 2 we evaluate the quality of results in the constraint-counting approach. The results show that the algorithm preforms very well compared to the optimum. We run the heuristic algorithm 40 times, each time with a different random order to the variables (recall

that the solution for the case of a single constraints combines a binary search on the values of each variable with a decision as to which variable to have as the outermost one, assigning to the other variables external values) and calculate, f_{min} and f_{max} in each iteration. We evaluate the assignments by checking how many constraints they satisfy, and output the best assignment. The running time of the algorithm that is shown in the table is the total running time, which is still negligible compared to the optimal algorithm's running time. In order to find the optimal assignment (solve the NP-hard problem), we go over all the assignments that use values in the restricted search space, hence the very high running times.

Query	$\|\mathcal{C}\|$	$\|\mathcal{X}\|$	algorithm		optimum	
			# of constraints satisfied	running time	# of constraints satisfied	running time
φ_1	8	4	5	3	6	35
φ_2	6	4	3	9	3	151
φ_3	8	4	2	42	2	202
φ_4	4	5	2	12	2	260
φ_5	11	8	2	29	6	620

Table 2. Evaluating the heuristic in the constraint-counting approach.

As seen in the table yak yak

We continue to evaluate the heuristic in the distance-minimization approach. Although we have no optimum to compare to, we believe that the distances, as shown in Table 3, are rather small. In order to calculate the distances, for an assignment f, a query φ, and a set of constraints $\mathcal{C}$, we use $dist(\varphi, f, \mathcal{C}) = \sqrt{\sum_{\langle \pi, c \rangle \in \mathcal{C}} (\llbracket \pi, \varphi^f \rrbracket - c)^2}$. Similar to the previous experiment, we run the algorithm 40 times, and we find three points: the minimal, maximal, and center of gravity (see Section **??**). The table shows that there is no clear winner between these three options. Since there is no optimum to compare to, we perform a sanity check as follows. We partitioned the constraints into two sets: $\mathcal{C}_1$ and $\mathcal{C}_2$. We find an assignment f_1 using the constraints $\mathcal{C}_1$. Then, we evaluate the assignments on the constraints $\mathcal{C}_2$, i.e., we calculate $dist(\varphi, f_1, \mathcal{C}_2)$, and finally, we perform the dual process. The table shows the two distances for the three mechanisms of generating an assignment.

Yak yak as seen.

1. Something about the lengths of the computations.

2. In the description of the sanity check – missing what was done in the "evaluate the assignments on the constraints $\mathcal{C}_2$"

Query	$\|\mathcal{C}\|$	$\|\mathcal{X}\|$	Distance			Sanity checks					
			minimum	maximum	center of gravity	minimum		maxmimum		center of gravity	
φ_1	8	4	0.212	0.166	0.189	0.250	0.180	0.111	0.304	0.320	0.226
φ_2	6	4	1.13	1.077	0.672	0.812	0.824	0.9	0.648	0.812	0.850
φ_3	8	4	0.165	0.187	0.780	0.234	0.212	0.578	0.111	0.9	0.465
φ_4	4	5	0.7	0.7	0.428	0.877	1.326	0.223	0.616	0.357	0.231
φ_5	11	8	0.845	0.357	0.6	0.792	0.323	0.670	0.194	0.908	0.173

Table 3. Evaluating the heuristic in the distance-minimization approach.

References

1. S. Almagor, U. Boker, and O. Kupferman. Discounting in LTL. 2012. submitted.
2. S. Almagor, U. Boker, and O. Kupferman. Formalizing and reasoning about quality. 2012. submitted.
3. R. Alur, T.A. Henzinger, and M.Y. Vardi. Parametric real-time reasoning. In *Proc. 25th ACM Symp. on Theory of Computing*, pages 592–601, 1993.
4. G. Ammons, R. Bodík, and J. R. Larus. Mining specifications. In *POPL*, pages 4–16, 2002.
5. G. Ammons, D. Mandelin, R. Bodk, and J.R. Larus. Debugging temporal specifications with concept analysis. In *Proc. ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation*, pages 182–195. Springer, 2003.
6. D. Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75(2):87–106, 1987.
7. G. Avni and O. Kupferman. Making weighted containment feasible: A heuristic based on simulation and abstraction. In *Proc. 23rd Int. Conf. on Concurrency Theory*, pages 84–99, 2012.
8. R. Bloem, R. Cavada, I. Pill, M. Roveri, and A. Tchaltsev. RAT: A tool for the formal analysis of requirements. In *Proc. 19th Int. Conf. on Computer Aided Verification*, volume 4590 of *Lecture Notes in Computer Science*, pages 263–267. Springer, 2005.
9. U. Boker, K. Chatterjee, T.A. Henzinger, and O. Kupferman. Temporal specifications with accumulative values. In *Proc. 26th IEEE Symp. on Logic in Computer Science*, pages 43–52, 2011.
10. W. Chan. Temporal-logic queries. In *Proc. 12th Int. Conf. on Computer Aided Verification*, volume 1855 of *Lecture Notes in Computer Science*, pages 450–463. Springer, 2000.
11. M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Patterns in property specifications for finite-state verification. In *ICSE*, pages 411–420, 1999.
12. B. Jobstmann and R. Bloem. Optimizations for ltl synthesis. In *FMCAD*, pages 117–124, 2006.
13. B. Jobstmann, S. Galler, M. Weiglhofer, and R. Bloem. Anzu: A tool for property synthesis. In *Proc. 19th Int. Conf. on Computer Aided Verification*, volume 4590 of *Lecture Notes in Computer Science*, pages 258–262, 2007.
14. B. Jobstmann, A. Griesmayer, and R. Bloem. Program repair as a game. In *Proc. 17th Int. Conf. on Computer Aided Verification*, volume 3576 of *Lecture Notes in Computer Science*, pages 226–238, 2005.
15. O. Kupferman. Sanity checks in formal verification. In *Proc. 17th Int. Conf. on Concurrency Theory*, volume 4137 of *Lecture Notes in Computer Science*, pages 37–51. Springer, 2006.
16. M.Z. Kwiatkowska. Quantitative verification: models techniques and tools. In *ESEC/SIGSOFT FSE*, pages 449–458, 2007.
17. P. Madhusudan. Learning algorithms and formal verification (invited tutorial). In *VMCAI*, page 214, 2007.
18. S. Moon, K. Lee, and D. Lee. Fuzzy branching temporal logic. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 34(2):1045–1055, 2004.
19. A. Pnueli. The temporal semantics of concurrent programs. *Theoretical Computer Science*, 13:45–60, 1981.
20. PROSYD. The Prosyd project on property-based system design. http://www.prosyd.org, 2007.
21. M. Roveri. Novel techniques for property assurance. Technical report, PROSYD FP6-IST-507219, 2007.
22. A. Solar-Lezama, R.M. Rabbah, R. Bodík, and K. Ebcioglu. Programming by sketching for bit-streaming programs. In *PLDI*, pages 281–294, 2005.