Formally Reasoning About Quality

SHAULL ALMAGOR, The Hebrew University
UDI BOKER, The Interdisciplinary Center
ORNA KUPFERMAN, The Hebrew University

In recent years, there has been a growing need and interest in formally reasoning about the quality of software and hardware systems. As opposed to traditional verification, in which one considers the question of whether a system satisfies a given specification or not, reasoning about quality addresses the question of *how well* the system satisfies the specification. We distinguish between two approaches to specifying quality. The first, *propositional quality*, extends the specification formalism with propositional quality operators, which prioritize and weight different satisfaction possibilities. The second, *temporal quality*, refines the "eventually" operators of the specification formalism with discounting operators, whose semantics takes into an account the delay incurred in their satisfaction.

In this article, we introduce two quantitative extensions of Linear Temporal Logic (LTL), one by propositional quality operators and one by discounting operators. In both logics, the satisfaction value of a specification is a number in [0, 1], which describes the quality of the satisfaction. We demonstrate the usefulness of both extensions and study the decidability and complexity of the decision and search problems for them as well as for extensions of LTL that combine both types of operators.

Categories and Subject Descriptors: F.4.1 [**Mathematical Logic**]: Temporal Logic

General Terms: Verification

Additional Key Words and Phrases: Automata, quality, LTL, model checking, synthesis

ACM Reference Format:
Shaull Almagor, Udi Boker, and Orna Kupferman. 2016. Formally reasoning about quality. J. ACM 63, 3, Article 24 (June 2016), 56 pages.
DOI: http://dx.doi.org/10.1145/2875421

1. INTRODUCTION

One of the main obstacles to the development of complex hardware and software systems lies in ensuring their correctness. A successful paradigm addressing this obstacle is *temporal-logic model checking*: given a mathematical model of the system and a temporal-logic formula that specifies a desired behavior of it, decide whether the model satisfies the formula [Clarke et al. 1999]. Correctness is Boolean: a system can either satisfy its specification or not satisfy it. The richness of today's systems, however, justifies specification formalisms that are *quantitative*. The quantitative setting arises directly in systems with quantitative aspects (multivalued/probabilistic/fuzzy) [Moon

The article combines and extends [Almagor et al. 2013] and [Almagor et al. 2014]. The research has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013)/ERC grant agreement no 278410.
Authors' addresses: S. Almagor, 18 Hamagid st. Jerusalem, Israel; email: shaull.almagor@mail.huji.ac.il; U. Boker, School of Computer Science, Interdisciplinary Center, Herzliya, Israel; email: udiboker@idc.ac.il; O. Kupferman, School of Computer Science and Engineering, Hebrew University, Jerusalem, Israel; email: orna@cs.huji.ac.il.

DOI: http://dx.doi.org/10.1145/2875421

et al. 2004; Kwiatkowska 2007; Faella et al. 2008; Droste and Rahonis 2009; Droste and Vogler 2012], but is applied also with respect to Boolean systems, in which it origins from the semantics of the specification formalism itself [De Alfaro et al. 2005; Almagor et al. 2013b].

When considering the *quality* of a system, satisfying a specification should no longer be a yes/no matter. Different ways of satisfying a specification should induce different levels of quality, which should be reflected in the output of the verification procedure. Consider, for example, the specification $\psi = \mathsf{G}(\mathit{request} \rightarrow \mathsf{F}(\mathit{response\_grant} \vee \mathit{response\_deny}))$; that is, every request is eventually responded to, with either a grant or a denial. When we evaluate ψ, there should be a difference between a computation that satisfies it with responses generated soon after requests and one that satisfies it with long waits. Moreover, there should be a difference between grant and deny responses, or cases in which no request is issued.

The issue of generating high-quality hardware and software systems attracts a lot of attention [Kan 2002; Spinellis 2003]. Quality, however, is traditionally viewed as an art or as an amorphous ideal. In this article, we suggest a framework for formalizing and reasoning about quality. Our working assumption is that different ways of satisfying a specification should induce different levels of quality, which should be reflected in the output of the verification procedure. Using our approach, a user can specify quality formally, according to the importance the user gives to components such as security, maintainability, runtime, delays, and more, then can formally reason about the quality of hardware and software systems.

1.1. Our Contribution

Recall the earlier specification example $\psi = \mathsf{G}(\mathit{request} \rightarrow \mathsf{F}(\mathit{response\_grant} \vee \mathit{response\_deny}))$. As the specification demonstrates, one can distinguish between two aspects of the quality of satisfaction. The first, which we call "propositional quality," concerns prioritizing related components of the specification. The second, which we call "temporal quality," concerns the waiting time to satisfaction of eventualities. We study both aspects, as well as their combination. Here, we describe the two aspects, along with our contribution in detail.

1.1.1. Propositional Quality. Quality is a rather subjective issue. Technically, we can talk about the quality of satisfaction of specifications since there are different ways to satisfy specifications. We introduce and study the linear temporal logic LTL[$\mathcal{F}$], that extends LTL with an arbitrary set $\mathcal{F}$ of functions over $[0, 1]$. Using the functions in $\mathcal{F}$, a specifier can formally and easily prioritize the different ways of satisfaction. The logic LTL[$\mathcal{F}$] is really a family of logics, each parameterized by a set $\mathcal{F} \subseteq \{f : [0, 1]^k \rightarrow [0, 1] \mid k \in \mathbb{N}\}$ of functions (of arbitrary arity) over $[0, 1]$. For example, $\mathcal{F}$ may contain the $\min\{x, y\}$, $\max\{x, y\}$, and $1-x$ functions, which are the standard quantitative analogues of the $\wedge$, $\vee$, and $\neg$ operators. As discussed in Section 1.2, such extensions to LTL have already been studied in the context of quantitative verification [Faella et al. 2008]. The novelty of LTL[$\mathcal{F}$], beyond its use in the specification of quality, is the ability to manipulate values by arbitrary functions. For example, $\mathcal{F}$ may contain the quantitative operator $\triangledown_\lambda$, for $\lambda \in [0, 1]$, that tunes down the quality of a subspecification. Formally, the quality of the satisfaction of the specification $\triangledown_\lambda \varphi$ is the multiplication of the quality of the satisfaction of φ by λ. Another useful operator is the weighted-average function $\oplus_\lambda$. There, the quality described by the formula $\varphi \oplus_\lambda \psi$ is the weighted (according to λ) average between the quality of φ and that of ψ. This enables the quality of the system to be an interpolation of different aspects of it. As an example, consider the formula $\mathsf{G}(\mathit{req} \rightarrow (\mathit{grant} \oplus_{\frac{3}{4}} \mathsf{X}\mathit{grant}))$. The formula specifies the fact that we want requests to be granted immediately and the grant to hold for two transactions. When this always holds, the satisfaction value is 1, corresponding to full satisfaction. We are quite content

with grants that are given immediately and last for only one transaction, in which case the satisfaction value is $\frac{3}{4}$, and less content when grants arrive with a delay, in which case the satisfaction value is $\frac{1}{4}$.

An LTL[$\mathcal{F}$] formula maps computations to a value in [0, 1]. We accordingly generalize classical decision problems—such as model checking, satisfiability, synthesis, and equivalence—to their quantitative analogues, which are search or optimization problems. For example, the equivalence problem between two LTL[$\mathcal{F}$] formulas φ_1 and φ_2 seeks the supremum of the difference in the satisfaction values of φ_1 and φ_2 over all computations. Of special interest is the extension of the synthesis problem. In conventional synthesis algorithms, we are given a specification to a reactive system, typically by means of an LTL formula, and we transform it into a system that is guaranteed to satisfy the specification with respect to all environments [Pnueli and Rosner 1989]. Little attention has been paid to the quality of the systems that are automatically synthesized[1]. Current efforts to address the quality challenge are based on enriching the game that corresponds to synthesis to a weighted one [Bloem et al. 2009; Černỳ et al. 2011]. Using LTL[$\mathcal{F}$], we are able to embody quality within the specification, which is very convenient.

In the Boolean setting, the automata-theoretic approach has proven to be very useful in reasoning about LTL specifications. The approach is based on translating LTL formulas to nondeterministic Büchi automata on infinite words [Vardi and Wolper 1986]. In the quantitative approach, it seems natural to translate formulas to *weighted automata* [Mohri 1997; Droste et al. 2009]. However, these extensively studied models are complicated and many problems become undecidable for them (e.g., the universality problem [Krob 1994; Almagor et al. 2011]). We show that we can use the approach taken in Faella et al. [2008], bound the number of possible satisfaction values of LTL[$\mathcal{F}$] formulas, and use this bound in order to translate LTL[$\mathcal{F}$] formulas to Boolean automata. From a technical point of view, the big challenge in our setting is to maintain the simplicity and the complexity of the algorithms for LTL, even though the number of possible values is exponential. We do so by restricting attention to feasible combinations of values assigned to the different subformulas of the specification. Essentially, our translation extends the construction of Vardi and Wolper [1986] by associating states of the automaton with functions that map each subformula to a satisfaction value. Using the automata-theoretic approach, we solve the verification and synthesis problems for LTL[$\mathcal{F}$] within the same complexity classes as the corresponding problems in the Boolean setting (as long as the functions in $\mathcal{F}$ are computable within these complexity classes; otherwise, they become the computational bottleneck). Our approach thus enjoys the fact that traditional automata-based algorithms are susceptible to well-known optimizations and symbolic implementations. It can also be easily implemented in existing tools.

1.1.2. Temporal Quality. In temporal quality, we consider delays as the factor that affects the quality of satisfaction. That is, the delay with which eventualities are satisfied determines the satisfaction value. One may try to reduce "temporal quality" to "propositional quality" using the fact that an eventuality involves a repeated choice between satisfying it in the present or delaying its satisfaction to the strict future. This attempt, however, requires unboundedly many applications of the propositional choice, and is similar to a repeated use of the X ("next") operator rather than a use of eventuality operators. Repeated use of X is a limited solution, as it partitions the future into finitely many zones, all of which are in the "near future," except for a single, unbounded, "far future." A more involved approach to distinguish between

[1]Note that we do not refer here to the challenge of generating optimal (say, in terms of state space) systems, but rather to quality measures that refer to how the specification is satisfied.

the "near" and "far" future includes bounded (prompt) eventualities [Bojańczyk and Colcombet 2006; Almagor et al. 2010]. There, one distinguishes between eventualities whose waiting time is bounded and ones that have no bound.

The weakness of both approaches is not surprising: correctness of LTL is Boolean, thus has an inherent dichotomy between satisfaction and dissatisfaction. The distinction between "near" and "far," however, is not dichotomous.

This suggests that, in order to formalize temporal quality, one must extend LTL to an unbounded setting. Realizing this, researchers have suggested augmenting temporal logics with *future discounting* [De Alfaro et al. 2003]. In the discounted setting, the satisfaction value of specifications is a numerical value, and depends, according to some discounting function, on the time waited for eventualities to get satisfied.

We introduce and study the linear temporal logic $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$—an augmentation by discounting of LTL. The logic $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$ is actually a family of logics, each parameterized by an arbitrary set $\mathcal{D}$ of discounting functions, namely, by a set of strictly decreasing functions from $\mathbb{N}$ to $[0, 1]$ that tend to 0 (e.g., linear decaying and exponential decaying). $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$ includes a discounting-"until" (U_η) operator, parameterized by a function $\eta \in \mathcal{D}$. We solve the model-checking threshold problem for $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$: given a Kripke structure $\mathcal{K}$, an $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$ formula φ, and a threshold $t \in [0, 1]$, the algorithm decides whether the satisfaction value of φ in $\mathcal{K}$ is at least t.

Similar to the case of $\mathrm{LTL}[\mathcal{F}]$, an attempt to handle $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$ with weighted automata involves problems that are, in general, undecidable [Almagor et al. 2011]. Unlike $\mathrm{LTL}[\mathcal{F}]$, the range of possible satisfaction values of an $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$ formula is infinite. Consequently, the border of decidability is different and the use of Boolean automata in reasoning about $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$ is much more challenging. Nevertheless, we show that, for threshold problems, we can translate $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$ formulas into (Boolean) nondeterministic Büchi automata (NBA), with the property that the automaton accepts a lasso computation if and only if the formula attains a value above the threshold on that computation. Our algorithm relies on the fact that the language of an automaton is nonempty if and only if there is a lasso witness for the nonemptiness. We cope with the infinitely many possible satisfaction values by using the discounting behavior of the eventualities and the given threshold in order to partition the state space into a finite number of classes. The complexity of our algorithm depends on the discounting functions used in the formula. We show that, for $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$ with standard discounting functions, such as exponential decaying (denoted by $\mathrm{LTL}^{\mathrm{disc}}[E]$), the problem is PSPACE-complete—not more complex than standard LTL. The fact that our algorithm uses Boolean automata also enables us to suggest a solution for threshold satisfiability, and to give a partial solution to threshold synthesis. In addition, it enables an adoption of heuristics and tools that exist for Boolean automata.

We note that, unlike the case of $\mathrm{LTL}[\mathcal{F}]$, the fact that in $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$ there are infinitely many satisfaction values implies that solving the threshold decision problems does not yield a solution to the search problem. A solution for the search problems remains elusive. As we show, however, the solution for the threshold problems does imply an approximate solution for the search problems, with every desired accuracy.

1.1.3. Combination of the Two Aspects and Other Extensions. After introducing $\mathrm{LTL}[\mathcal{F}]$ and $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$ and studying their decision problems, we augment $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$ with propositional quality operators. Beyond the operators min, max, and $\neg$, which are already present, two basic propositional quality operators of $\mathrm{LTL}[\mathcal{F}]$ are the multiplication of an $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$ formula by a constant in $[0, 1]$, and the averaging between the satisfaction values of two $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$ formulas. We show that, while the first

extension does not increase the expressive power of $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$ or its complexity, the latter causes the validity and model-checking problems to become undecidable. In fact, things become undecidable even if we allow averaging in combination with a single discounting function. Recall that this is in contrast with the extension of discounted CTL with an average operator, in which the complexity of the model-checking problem stays polynomial [De Alfaro et al. 2005].

We consider additional extensions of $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$. First, we study a variant of the discounting-eventually operators, in which we allow the discounting to tend to arbitrary values in $[0, 1]$ (rather than to 0). This captures the intuition that we are not always pessimistic about the future, but can be, for example, ambivalent about it, by tending to $\frac{1}{2}$. We show that all our results hold under this extension. Second, we add to $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$ *past* operators and their discounting versions (specifically, we allow a discounting-"since" operator, and its dual). In the traditional semantics, past operators enable clean specifications of many interesting properties, make the logic exponentially more succinct, and can still be handled within the same complexity bounds [Lichtenstein et al. 1985; Laroussinie and Schnoebelen 1994]. We show that the same holds for the discounted setting. Finally, we show how $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$ and algorithms for it can also be used for reasoning about weighted systems.

1.2. Related Work

The quantitative setting has been an active area of research, providing many works on quantitative logics and automata [Kupferman and Lustig 2007; Droste et al. 2008; Droste and Rahonis 2009; Almagor and Kupferman 2011; Droste and Vogler 2012].

Conceptually, our work aims at formalizing quality, having a different focus from each of the other works. Technically, the main difference between our setting and most of the other approaches is the source of quantitativeness: There, it stems from the nature of the system (multivalued/quantitative/probabilistic/fuzzy), whereas in our setting it stems from the richness of the new functional and discounting operators.

The common practice in these works is to give a quantitative interpretation to the Boolean and temporal operators (e.g., by associating a conjunction with *min*) [Moon et al. 2004; Faella et al. 2008], for coping with the quantitative nature of the system. In our setting, on the other hand, the system may be either Boolean or quantitative, while the quality measures are obtained by functions that are much richer than the quantitative interpretation of the Boolean operators—all sets of functions over the $[0, 1]$ interval are allowed. We elaborate here on some of these works.

—Early work handles *multivalued systems*, in which the values of atomic propositions are taken from a finite domain,f or example, "uninitialized," "unknown," and "don't care" [IEEE 1993]. Beyond richer expressiveness, multivalues, sometimes arranged in a lattice, are useful in abstraction methods, query checking, and verification of systems from inconsistent viewpoints, in which multivalues are used to model missing, hidden, or varying information [Bruns and Godefroid 2004; Kupferman and Lustig 2007]. Satisfaction of formulas is then multivalued, corresponding to the values of atomic propositions.

—More sophisticated multivalued systems consider *real-valued signals* [Donzé and Maler 2010]. The quantitative setting there stems from both time intervals and predicates over the value of atomic propositions. In particular, signal temporal logic [Donzé et al. 2012] allows a quantitative reference to the temporal distance between events. The motivation and type of questions studied in this setting, however, is different from ours.

—In *fuzzy* temporal logic, for example, Moon et al. [2004], formulas are interpreted over fuzzy systems—ones in which the occurrence of an event or of a state at a certain point in time gets a value in [0, 1]. Such systems and reasoning about them have also been studied in the context of quantitative verification [De Alfaro et al. 2004; Faella et al. 2008]. Formulas are satisfied with a value in [0, 1], following the fuzziness of events and states.
—*Probabilistic* temporal logic is interpreted over Markov chains and Markov decision processes [Hansson and Jonsson 1994; Desharnais et al. 2004; Kwiatkowska 2007]. Each transition in the system has a value in [0, 1], denoting the probability of taking it. Accordingly, one can talk about the probability of reaching a state or satisfying a desired property. A formula then gets a value between 0 and 1, indicating the probability of satisfying it.

In the temporal-quality front, the notion of discounting has been studied in several fields, such as economy, game-theory, and Markov decision processes [Shapley 1953]. In the area of formal verification, it was suggested in De Alfaro et al. [2003] to augment the μ-calculus with discounting operators. The discounting suggested there is exponential; that is, with each iteration, the satisfaction value of the formula decreases by a multiplicative factor in $(0, 1]$. Algorithmically, De Alfaro et al. [2003] show how to evaluate discounted μ-calculus formulas with arbitrary precision. Formulas of LTL can be translated to the μ-calculus [De Alfaro et al. 2003], thus can be used in order to approximately model-check discounted-LTL formulas. However, the translation from LTL to the μ-calculus involves an exponential blowup [Dam 1994] (and is complicated), making this approach inefficient. Moreover, our approach allows for arbitrary discounting functions, and the algorithm returns an exact solution to the threshold model-checking problem, which is more difficult than the approximation problem.

A different approach to temporal quality is to consider *mean-payoff* semantics. Then, the future can have a strong effect on the value of a computation, providing that it is persistent and small perturbations are averaged out. Works in this direction include Boker et al. [2014], in which LTL is augmented with Boolean assertions over the sum and average of quantitative atomic assertions, and Bohy et al. [2013], in which atomic propositions are assigned weights and the goal is to synthesize, given an LTL formula φ, a transducer that realizes φ and such that the mean-payoff of the output letters in every computation is above some threshold. Bouyer et al. [2014] augment LTL with averaging temporal operators, and study the related satisfiability and model-checking problems. It is shown that all variants of the problems become undecidable, and connections are drawn to our work.

Closer to our work is De Alfaro et al. [2005], in which CTL is augmented with discounting and weighted-average operators. The motivation in De Alfaro et al. [2005] is to introduce a logic whose semantics is not too sensitive to small perturbations in the model. Accordingly, formulas are evaluated on weighted systems or on Markov chains. Adding discounting and weighted-average operators to CTL preserves its appealing complexity, and the model-checking problem for the augmented logic can be solved in polynomial time. As is the case in the traditional Boolean semantics, the expressive power of discounted CTL is limited.

Perhaps closest to our approach is Mandrali [2012], in which a version of discounted-LTL was introduced. Semantically, there are two main differences between the logics. The first is that [Mandrali 2012] uses a discounted sum, while we interpret discounting without accumulation. The second is that the discounting there replaces the standard temporal operators; thus, all eventualities are discounted. As discounting functions tend to 0, this strictly restricts the expressive power of the logic, and one cannot specify traditional eventualities in it. On the positive side, it enables a clean algebraic

characterization of the semantics; the contribution in Mandrali [2012] is a comprehensive study of the mathematical properties of the logic. Yet, Mandrali [2012] does not study algorithmic questions about the logic. We, on the other hand, focus on the algorithmic properties of the logic, specifically on the model-checking problem. In addition, we are the first to combine the two aspects.

More recently, we see works that use our formalism and approach for reasoning about quality. Fränzle et al. [2015] suggest an extension of $\text{LTL}^{\text{disc}}[E]$ to continuous-time computations. In the more applicative front, Nakagawa and Hasuo [2015] had recently used our framework in order to solve the synthesis problem with a near-optimal quality. The authors cope with the infinite range of satisfaction values of $\text{LTL}^{\text{disc}}[\mathcal{D}]$ by partitioning the infinite range into finitely many sections whose number depends on a given allowed error factor.

Finally, in Almagor et al. [2013a], we automatically generate the factors in $\text{LTL}^{\triangledown}$ formulas according to samples of computations and corresponding satisfaction values. In Almagor and Kupferman [2015], we study LTL[$\mathcal{F}$] synthesis in a stochastic setting, in which we maximize the expected satisfaction value of a synthesized transducer.

2. FORMALIZING PROPOSITIONAL QUALITY

In this section, we introduce the temporal logic LTL[$\mathcal{F}$] (Section 2.1) and the verification and synthesis questions that arise once we add quality measures (Section 2.2). We then provide some observations on LTL[$\mathcal{F}$] (Section 2.3), followed by an algorithm for translating a given LTL[$\mathcal{F}$] formula into an automaton (Section 2.4). This automata-theoretic approach will be our main tool in solving decision and search problems for LTL[$\mathcal{F}$] (Section 2.5).

2.1. The Temporal Logic LTL[$\mathcal{F}$]

The linear temporal logic LTL[$\mathcal{F}$] generalizes LTL by replacing the Boolean operators of LTL with arbitrary functions over $[0, 1]$. The logic is actually a family of logics, each parameterized by a set $\mathcal{F}$ of functions.

Syntax. Let AP be a set of Boolean atomic propositions, and let $\mathcal{F} \subseteq \{f : [0,1]^k \to [0,1] \mid k \in \mathbb{N}\}$ be a set of functions over $[0, 1]$. Note that the functions in $\mathcal{F}$ may have different arities. An LTL[$\mathcal{F}$] formula is one of the following:

—`True`, `False`, or p, for $p \in AP$.
—$f(\varphi_1, \ldots, \varphi_k)$, $\mathsf{X}\varphi_1$, or $\varphi_1 \mathsf{U} \varphi_2$, for LTL[$\mathcal{F}$] formulas $\varphi_1, \ldots, \varphi_k$ and a function $f \in \mathcal{F}$.

We define the description size $|\varphi|$ of an LTL[$\mathcal{F}$] formula φ to be the number of nodes in the generating tree of φ. Note that the function symbols in $\mathcal{F}$ are treated as constant-length symbols.

Semantics. We define the semantics of LTL[$\mathcal{F}$] formulas with respect to infinite computations over AP. In Section 3.2, we also consider LTL[$\mathcal{F}$] over finite computations. A *computation* is a word $\pi = \pi_0, \pi_1, \ldots \in (2^{AP})^\omega$. We use π^i to denote the suffix $\pi_i, \pi_{i+1}, \ldots$. The semantics maps a computation π and an LTL[$\mathcal{F}$] formula φ to the *satisfaction value* of φ in π, denoted $[\![\pi, \varphi]\!]$. The satisfaction value is defined inductively as described in Table I.[2]

Note that the satisfaction value of $\varphi_1 \mathsf{U} \varphi_2$ in π is obtained by going over all suffixes of π, searching for a position $i \geq 0$ that maximizes the minimum between the satisfaction

[2]The observant reader may be concerned by our use of max and min in contexts in which sup and inf are in order. In Lemma 2.5, we prove that there are only finitely many satisfaction values for a formula φ, thus the semantics is well defined.

Table I. The Semantics of LTL[$\mathcal{F}$]

Formula	Satisfaction value
$[\![\pi, \texttt{True}]\!]$	1
$[\![\pi, \texttt{False}]\!]$	0
$[\![\pi, p]\!]$	1 if $p \in \pi_0$ 0 if $p \notin \pi_0$
$[\![\pi, f(\varphi_1, \ldots, \varphi_k)]\!]$	$f([\![\pi, \varphi_1]\!], \ldots, [\![\pi, \varphi_k]\!])$
$[\![\pi, \mathsf{X}\varphi_1]\!]$	$[\![\pi^1, \varphi_1]\!]$
$[\![\pi, \varphi_1 \mathsf{U} \varphi_2]\!]$	$\max_{i \geq 0}\{\min\{[\![\pi^i, \varphi_2]\!], \min_{0 \leq j < i} [\![\pi^j, \varphi_1]\!]\}\}$

value of φ_2 in π^i (i.e., the satisfaction value of the eventuality) and all the satisfaction values of φ_1 in π^j for $0 \leq j < i$ (i.e., the satisfaction value of φ_1 until the eventuality is taken into account).

It is not hard to prove by induction on the structure of the formula that, for every computation π and formula φ, it holds that $[\![\pi, \varphi]\!] \in [0, 1]$.

The logic LTL coincides with the logic LTL[$\mathcal{F}$] for $\mathcal{F}$ that corresponds to the usual Boolean operators. For simplicity, we use these operators as an abbreviation for the corresponding functions, as described later. In addition, we introduce notations for some useful functions. Let $x, y \in [0, 1]$ be satisfaction values and $\lambda \in [0, 1]$ be a parameter. Then,

- $\neg x = 1 - x$
- $x \vee y = \max\{x, y\}$
- $x \wedge y = \min\{x, y\}$
- $x \rightarrow y = \max\{1 - x, y\}$
- $\bigtriangledown_\lambda x = \lambda \cdot x$
- $x \oplus_\lambda y = \lambda \cdot x + (1 - \lambda) \cdot y$

To see that LTL indeed coincides with LTL[$\mathcal{F}$] for $\mathcal{F} = \{\neg, \vee, \wedge\}$, note that, for this $\mathcal{F}$, all formulas are mapped to $\{0, 1\}$ in a way that agrees with the semantics of LTL. In particular, observe that, under these notations, we can write the semantics of $[\![\pi, \varphi_1 \mathsf{U} \varphi_2]\!]$ as $\bigvee_{i \geq 0}([\![\pi^i, \varphi_2]\!] \wedge \bigwedge_{0 \leq j < i} [\![\pi^j, \varphi_1]\!])$, which coincides with the semantics of LTL.

Other useful abbreviations are the "eventually" and "always" temporal operators, defined as follows.

—$\mathsf{F}\varphi_1 = \texttt{True}\mathsf{U}\varphi_1$. Thus, $[\![\pi, \mathsf{F}\varphi_1]\!] = \max_{i \geq 0}\{[\![\pi^i, \varphi_1]\!]\}$.

—$\mathsf{G}\varphi_1 = \neg\mathsf{F}\neg\varphi_1$. Thus, $[\![\pi, \mathsf{G}\varphi_1]\!] = \min_{i \geq 0}\{[\![\pi^i, \varphi_1]\!]\}$.

Kripke Structures and Transducers. We model closed systems by Kripke structures and open systems by transducers, both generating infinite computations.

A *Kripke structure* is a tuple $\mathcal{K} = \langle AP, S, I, \rho, L\rangle$, where AP is a finite set of atomic propositions, S is a finite set of states, $I \subseteq S$ is the set of initial states, $\rho \subseteq S \times S$ is a total transition relation, and $L : S \rightarrow 2^{AP}$ is a labeling function. A *trace* of $\mathcal{K}$ is a sequence $s = s_0, s_1, \ldots$ of states such that $s_0 \in I$; for all $0 \leq j$, it holds that $\langle s_j, s_{j+1}\rangle \in \rho$. A word $\pi = \pi_0, \pi_1, \ldots$ over 2^{AP} is a *computation* of $\mathcal{K}$ if there exists a trace s of $\mathcal{K}$ such that $\pi_j = L(s_j)$ for all $0 \leq j$.

In the Boolean setting of LTL, a Kripke structure satisfies a formula φ if all its computations satisfy the formula. Adopting this universal approach, the satisfaction value of an LTL[$\mathcal{F}$] formula φ in a Kripke structure $\mathcal{K}$, denoted $[\![\mathcal{K}, \varphi]\!]$, is induced by the "worst" computation of $\mathcal{K}$, namely, the one in which φ has the minimal satisfaction value. Formally,[3] $[\![\mathcal{K}, \varphi]\!] = \min\{[\![\pi, \varphi]\!] : \pi \text{ is a computation of } \mathcal{K}\}$.

[3]Since a Kripke structure may have infinitely many computations, here, too, we should have *a priori* used inf; the use of min is justified by Lemma 2.5.

In the setting of open systems, the set of atomic propositions is partitioned into sets I and O of input and output signals. An (I, O)-transducer then models the computations generated (deterministically) by the system when it interacts with an environment that generates infinite sequences of input signals. Formally, an (I, O)-*transducer* is a tuple $\mathcal{T} = \langle I, O, S, s_0, \rho, L\rangle$, where I and O are finite sets of input and output signals, respectively, S is a finite set of states, $s_0 \in S$ is an initial state, $\rho : S \times 2^I \to S$ maps a state and an assignment of the input signals to a successor state, and $L : S \to 2^O$ is a labeling function. Every sequence $i = i_0, i_1, \ldots \in (2^I)^\omega$ of assignments of the input signals induces a single trace $s = s_0, s_1, \ldots$ of $\mathcal{T}$, satisfying $s_{j+1} = \rho(s_j, i_j)$ for all $j \geq 0$, and induces the computation $\pi = \pi_0, \pi_1, \ldots$ over $2^{I \cup O}$ in which $\pi_j = i_j \cup L(s_j)$ for all $j \geq 0$.

Examples. We demonstrate the usefulness of LTL[$\mathcal{F}$] with some examples. In Examples 2.1 and 2.2, we utilize the $\oplus$ and $\triangledown$ operators. In Examples 2.3 and 2.4, we utilize more complex functions.

Example 2.1. Consider two servers performing in parallel the same task (e.g., sending messages). Server 1 is twice as fast as Server 2, accordingly sending twice as many messages. Assume that the LTL[$\mathcal{F}$] formulas φ_1 and φ_2 specify the quality of each of the servers. The quality of the system is then specified by the LTL[$\mathcal{F}$] formula $\varphi_1 \oplus_{\frac{2}{3}} \varphi_2$.

Example 2.2. Consider a scheduler that receives requests and generates grants. Consider the LTL[$\mathcal{F}$] formula $\mathsf{G}(req \rightarrow \mathsf{F}(grant \oplus_{\frac{1}{2}} \mathsf{X}grant)) \wedge \neg(\triangledown_{\frac{3}{4}} \mathsf{G}\neg req)$. The satisfaction value of the formula is 1 if every request is eventually granted; the grant lasts for two consecutive steps. If a grant holds only for a single step, then the satisfaction value is reduced to $\frac{1}{2}$. In addition, if there are no requests, then the satisfaction value is at most $\frac{1}{4}$. This shows how we can embed vacuity tests in the formula.

Example 2.3. Consider a server room, in which k servers perform a task. Every server emits heat. The function $f : [0, 1]^k \to [0, 1]$ describes the (normalized to $[0, 1]$) temperature of the room at a given time, as a function of each server's heat. Note that the temperature computation need not be a simple average function. Assume that ψ_i is an LTL[$\mathcal{F}$] formula specifying the heat generated by the ith heater. Note that ψ_i may be Boolean (indicating whether the heater is on or off) or more accurate, and depends on the actual value of the heat. The latter can be specified either by nested LTL[$\mathcal{F}$] formulas or by weighted propositions[4]. Then, the LTL[$\mathcal{F}$] formula $\mathsf{F}(f(\psi_1, \ldots, \psi_n))$ specifies the highest heat of the room over the course of the servers' operation.

Example 2.4. Consider a mechanical arm that is supposed to lift objects from a surface (e.g., from a conveyor belt). The arm has k joints, and each joint has two possible angles. The joints determine the configuration of the arm, and the latter determines the quality of the grab operation – the more extended the arm is, the less accurate the action gets. Assume that ψ_i is an LTL[$\mathcal{F}$] formula specifying the configuration of the ith joint. As in the previous example, ψ_i need not be Boolean (e.g., it may be its angle, or it may be a nested LTL[$\mathcal{F}$] formula, such as $retracted \oplus_{\frac{5}{6}} \mathsf{X}retracted$, indicating that, ideally, the joint is retracted for two time units, with the first time unit being much more important). The LTL[$\mathcal{F}$] formula $\mathsf{G}(grab \rightarrow f(\psi_1, \psi_2, \ldots, \psi_k))$ specifies the quality (accuracy) of the grab operation, where f is a function that calculates the quality of the grabbing given the configuration of the k joints.

[4]As we show in Section 3, it is easy to extend our framework to handle such propositions.

2.2. The Search and Decision Questions

In the Boolean setting, an LTL formula maps computations to $\{\texttt{True}, \texttt{False}\}$. In the quantitative setting, an LTL[$\mathcal{F}$] formula maps computations to $[0, 1]$. Classical decision problems—such as model checking, satisfiability, synthesis, and equivalence—are accordingly generalized to their quantitative analogues, which are search or optimization problems. In this section, we specify these questions with respect to LTL[$\mathcal{F}$]. While the definition here focuses on LTL[$\mathcal{F}$], the questions can be asked with respect to arbitrary quantitative specification formalisms, with the expected adjustments.

—**Satisfiability and validity.** In the Boolean setting, the satisfiability problem asks, given an LTL formula φ, whether φ is satisfiable. In the quantitative setting, it asks what the optimal way to satisfy φ is. Thus, the *satisfiability* problem gets as input an LTL[$\mathcal{F}$] formula φ and returns $\sup\{[\![\pi, \varphi]\!] : \pi \text{ is a computation}\}$. Simultaneoulsy, the *validity* problem returns, given an LTL[$\mathcal{F}$] formula φ, the value $\inf\{[\![\pi, \varphi]\!] : \pi \text{ is a computation}\}$, describing the least favorable way to satisfy the specification. In the case of LTL[$\mathcal{F}$], Lemma 2.5 guarantees that there are only finitely many possible satisfaction values for φ. Accordingly, we can replace the sup and inf operators discussed earlier with max and min, respectively, and can also extend the satisfiability and validity problems to ones that return a computation π with which the maximal or minimal value is obtained.

—**Implication and equivalence.** In the Boolean setting, the implication problem asks, given two LTL formulas φ_1 and φ_2, whether every computation that satisfies φ_1 also satisfies φ_2. In the quantitative setting, it asks about the difference between the satisfaction values of φ_1 and φ_2. Thus, the *implication* problem gets as input two LTL[$\mathcal{F}$] formulas φ_1 and φ_2 and returns $\max\{[\![\pi, \varphi_1]\!] - [\![\pi, \varphi_2]\!] : \pi \text{ is a computation}\}$. As in the Boolean setting, we may consider the symmetric version of implication. Formally, the *equivalence* problem gets as input two LTL[$\mathcal{F}$] formulas φ_1 and φ_2 and returns

$$\max\{|[\![\pi, \varphi_1]\!] - [\![\pi, \varphi_2]\!]| : \pi \text{ is a computation}\}.$$

—**Model checking.** The model-checking problem is extended from the Boolean setting to find, given a system $\mathcal{K}$ and an LTL[$\mathcal{F}$] formula φ, the satisfaction value $[\![\mathcal{K}, \varphi]\!]$. In the Boolean setting, good model-checking algorithms return a counterexample to the satisfaction of the specification when it does not hold in the system. The quantitative counterpart is to return a computation π of K that satisfies φ in the least favorable way.

—**Realizability and synthesis.** In the Boolean setting, the realizability problem gets as input an LTL formula over $I \cup O$, for sets I and O of input and output signals, and asks for the existence of an (I, O)-transducer, all of whose computations satisfy the formula. In the quantitative analogue, we seek the generation of high-quality systems. Accordingly, given an LTL[$\mathcal{F}$] formula φ over $I \cup O$, the realizability problem is to find $\max\{[\![\mathcal{T}, \varphi]\!] : \mathcal{T} \text{ is an } (I, O)\text{-transducer}\}$. The synthesis problem is then to find a transducer that attains this value.[5]

Decision Problems. The questions presented earlier are search and optimization problems. It is sometimes interesting to consider the decision problems that they induce when referring to a specific threshold. For example, the model-checking decision problem is to decide, given a system $\mathcal{K}$, a specification φ, and a threshold t, whether $[\![\mathcal{K}, \varphi]\!] \geq t$. For some problems, there are natural thresholds to consider. For example, in the

[5]The specification of the problem does not require the transducer to be finite. As we shall show, however, as in the case of LTL, if some transducer that attains the value exists, there is also a finite-state one that does so.

implication problem, asking whether $\max\{[\![\pi, \varphi_1]\!] - [\![\pi, \varphi_2]\!] : \pi \text{ is a computation}\} \geq 0$ amounts to asking whether, for all computations π, we have that $[\![\pi, \varphi_1]\!] \geq [\![\pi, \varphi_2]\!]$, which indeed captures implication.

Working with strict vs. nonstrict inequality when considering threshold problems may clearly affect their answer. Moreover, it may also require a completely different approach for solving the problem. This is demonstrated in Sections 4.3 and 6.1.

2.3. Properties of LTL[$\mathcal{F}$]

Bounding the number of satisfaction values. For an LTL[$\mathcal{F}$] formula φ, let $V(\varphi) = \{[\![\pi, \varphi]\!] : \pi \in (2^{AP})^\omega\}$. That is, $V(\varphi)$ is the set of possible satisfaction values of φ in arbitrary computations. We first show that this set is finite for all LTL[$\mathcal{F}$] formulas.

LEMMA 2.5. *For every* LTL[$\mathcal{F}$] *formula* φ*, we have that* $|V(\varphi)| \leq 2^{|\varphi|}$.

PROOF. The proof proceeds by induction on the structure of φ. The base case has three possibilities: `True`, `False`, and $p \in AP$, in all of which $|\varphi| = 1$ and $|V(\varphi)| \leq 2$. For the induction step, we consider the following cases.

—Let $\varphi = f(\psi_1, \ldots, \psi_k)$ for $f \in \mathcal{F}$. The number of possible inputs for f is at most $\prod_{i=1}^{k} |V(\psi_i)|$. By the induction hypothesis, for every $1 \leq i \leq k$, we have that $|V(\psi_i)| \leq 2^{|\psi_i|}$. Thus, the number of possible inputs for f is at most $\prod_{i=1}^{k} 2^{|\psi_i|} = 2^{\sum_{i=1}^{k} |\psi_i|} \leq 2^{|\varphi|-1} < 2^{|\varphi|}$.
—Let $\varphi = \mathsf{X}\psi$. Then, $V(\varphi) = V(\psi)$, and the claim follows immediately from the induction hypothesis.
—Let $\varphi = \psi_1 \mathsf{U} \psi_2$. By the semantics of the operator U, the satisfaction value of φ is defined by means of max and min on the satisfaction values of ψ_1 and ψ_2. Hence, $V(\varphi) \subseteq V(\psi_1) \cup V(\psi_2)$, implying that $|V(\varphi)| \leq |V(\psi_1)| + |V(\psi_2)|$. By the induction hypothesis, the latter is at most $2^{|\psi_1|} + 2^{|\psi_2|} \leq 2^{|\psi_1|+|\psi_2|} = 2^{|\varphi|-1} < 2^{|\varphi|}$. □

The good news that follows from Lemma 2.5 is that every LTL[$\mathcal{F}$] formula has only finitely many possible satisfaction values. This enabled us to replace the sup and inf operators in the semantics by the more friendly max and min. It also implies that we can point to witnesses that exhibit the satisfaction values. The bad news is that Lemma 2.5 gives an exponential bound only to the number of satisfaction values. We now show that this exponential bound is tight.

Example 2.6. Consider the logic LTL[$\{\oplus\}$], augmenting LTL with the average function, where, for every $x, y \in [0, 1]$, we have that $x \oplus y = \frac{1}{2}x + \frac{1}{2}y$. Let $n \in \mathbb{N}$ and consider the formula $\varphi_n = p_1 \oplus (p_2 \oplus (p_3 \oplus (p_4 \oplus \ldots p_n))\ldots)$. The length of φ_n is in $O(n)$ and the nesting depth of $\oplus$ operators in it is n. For every computation π, it holds that

$$[\![\pi, \varphi_n]\!] = \frac{1}{2}[\![\pi_0, p_1]\!] + \frac{1}{4}[\![\pi_0, p_2]\!] + \cdots + \frac{1}{2^{n-1}}[\![\pi_0, p_{n-1}]\!] + \frac{1}{2^{n-1}}[\![\pi_0, p_n]\!].$$

Hence, every assignment $\pi_0 \subseteq \{p_1, \ldots, p_{n-1}\}$ to the first position in π induces a different satisfaction value for $[\![\pi, \varphi_n]\!]$, implying that there are 2^{n-1} different satisfaction values for φ_n.

2.3.1. Calculating $V(\varphi)$. Lemma 2.5 provides a way to compute $V(\varphi)$ by traversing the generating tree of φ: for every subformula ψ in the tree, compute $V(\psi)$ recursively, as in the proof of Lemma 2.5. This algorithm, however, takes exponential time as well as exponential space for writing all the values. While the exponential time is unavoidable

due to the exponential number of values, there is a more efficient procedure, space-wise, for enumerating $V(\varphi)$ in PSPACE, as follows.

For a formula φ, consider its generating tree T. A $\{0,1\}$-*labeling* of T is an assignment ℓ of a label in $\{0,1\}$ to every node in T that is either an atomic proposition (i.e., a leaf) or a node that corresponds to a $\psi_1 \mathsf{U} \psi_2$ formula. Given a $\{0,1\}$-labeling ℓ of T, the *evaluation* of T and ℓ is a value v that is computed recursively for each node ψ in T, as follows.

—If $\psi = \texttt{True}$ or $\psi = \texttt{False}$, then $v(\psi) = 1$ or $v(\psi) = 0$, respectively.
—If ψ is an atomic proposition, then $v(\psi) = \ell(\psi)$.
—If $\psi = f(\varphi_1, \ldots, \varphi_k)$, then $v = f(v(\varphi_1), \ldots, v(\varphi_k))$.
—If $\psi = \mathsf{X}\varphi$, then $v(\psi) = v(\varphi)$.
—If $\psi = \varphi_1 \mathsf{U} \varphi_2$, then $v(\psi) = \begin{cases} v(\varphi_1) & \text{if } \ell(\psi) = 0 \\ v(\varphi_2) & \text{if } \ell(\psi) = 1. \end{cases}$

It is easy to see (similar to Lemma 2.5) that every possible satisfaction value of φ is obtained as $v(\varphi)$ for some labeling ℓ of T. Also, every labeling of T induced a possible satisfaction value of φ. Thus, to iterate through the values in $V(\varphi)$, it is sufficient to iterate through all the possible labelings of T. Assume that the functions in $\mathcal{F}$ are computable in PSPACE. Then, since the description of a labeling is polynomial (indeed, linear) in the size of T, and since evaluating $v(\varphi)$ from a given labeling can be done in PSPACE, we get that iterating through all the values can be done in PSPACE. If the functions in $\mathcal{F}$ are not computable in PSPACE, then their computation becomes the computational bottleneck.

A Boolean look at LTL[$\mathcal{F}$]. The logic LTL[$\mathcal{F}$] provides means to generalize LTL to a quantitative setting. Yet, one may consider a Boolean logic whose atoms are Boolean assertions on the values of LTL[$\mathcal{F}$] formulas. For example, we can define a logic with atoms of the form $\varphi_1 \geq \varphi_2$ or $\varphi_1 \geq v$ for LTL[$\mathcal{F}$] formulas φ_1 and φ_2, and a value $v \in [0,1]$. It is then natural to compare the expressiveness and succinctness of such a logic with respect to LTL.

Intuitively, the role the functions in $\mathcal{F}$ play in LTL[$\mathcal{F}$] is propositional, in the sense that the functions do not introduce new temporal operators. We now formalize this intuition, showing that, for every LTL[$\mathcal{F}$] formula φ and predicate $P \subseteq [0,1]$, there exists an LTL formula $Bool(\varphi, P)$ asserting that the satisfaction value of φ is in P. Formally, we have the following.

THEOREM 2.7. *For every* LTL[$\mathcal{F}$] *formula* φ *and predicate* $P \subseteq [0,1]$*, there exists an* LTL *formula* $Bool(\varphi, P)$*, of length at most exponential in* φ*, such that, for every computation* $\pi \in (2^{AP})^\omega$*, it holds that* $[\![\pi, \varphi]\!] \in P$ *if and only if* $\pi \models Bool(\varphi, P)$.

PROOF. The proof is by induction on the structure of φ.

—$Bool(\texttt{True}, P) = \begin{cases} \texttt{True} & \text{if } 1 \in P \\ \texttt{False} & \text{if } 1 \notin P. \end{cases}$
—$Bool(\texttt{False}, P) = \begin{cases} \texttt{True} & \text{if } 0 \in P \\ \texttt{False} & \text{if } 0 \notin P. \end{cases}$
—For $p \in AP$, we have that $Bool(p, P) = \begin{cases} \texttt{True} & \text{if } 0 \in P \text{ and } 1 \in P \\ p & \text{if } 0 \notin P \text{ and } 1 \in P \\ \neg p & \text{if } 0 \in P \text{ and } 1 \notin P \\ \texttt{False} & \text{if } 0 \notin P \text{ and } 1 \notin P. \end{cases}$
—$Bool(f(\varphi_1, \ldots, \varphi_k), P) = \bigvee_{\{d_1 \in V(\varphi_1), \ldots, d_k \in V(\varphi_k):\ f(d_1, \ldots, d_k) \in P\}} Bool(\varphi_1, d_1) \wedge \cdots \wedge Bool(\varphi_k, d_k)$.

—$Bool(X\varphi_1, P) = \mathsf{X}Bool(\varphi_1, P)$.
—If $\varphi = \varphi_1 \mathsf{U} \varphi_2$, we decompose $Bool(\varphi, P)$ to a disjunction of the LTL formulas $Bool(\varphi, c)$ for $c \in P \cap V(\varphi)$, defined as follows.

$$Bool(\varphi_1 \mathsf{U} \varphi_2, c) = (Bool(\varphi_1, [c, 1]) \mathsf{U} Bool(\varphi_2, [c, 1])) \wedge \neg(Bool(\varphi_1, (c, 1]) \mathsf{U} Bool(\varphi_2, (c, 1])).$$

Intuitively, the first conjunct in $Bool(\varphi, c)$ guarantees that $[\![\pi, \varphi]\!] \geq c$, and the second part guarantees that $[\![\pi, \varphi]\!] \not> c$.

We note that, in the construction of $Bool(f(\varphi_1, \ldots, \varphi_k), P)$ and $Bool(\varphi_1 \mathsf{U} \varphi_2, P)$, it is often possible to use relations between the different values in $P \cap V(\varphi)$ and combine the disjuncts of different values or refrain from the restriction to a singleton P. □

The translation described in the proof of Theorem 2.7 may involve an exponential blowup. It is thus interesting to study whether this blow-up is unavoidable. One needs to tread carefully when studying the blowup, as there are two aspects to it. First, we can look at the size of a minimal Boolean formula that is equivalent to $Bool(\varphi, P)$, which we dub the *output complexity*. Second, we can consider the complexity of the procedure in Theorem 2.7, which translates φ and P to $Bool(\varphi, P)$, dubbed the *translation complexity*.

We start by showing that the output complexity can be exponential in $|\varphi|$. This stems from the fact that, in LTL[$\mathcal{F}$], we can represent every Boolean function $f : \{0, 1\}^n \to \{0, 1\}$ with the formula $f(p_1, \ldots, p_n)$, which we have defined to have length $n + 1$, whereas some Boolean functions require exponentially long propositional formulas [Shannon 1949], thus a formula equivalent to $Bool(f(p_1, \ldots, p_n), \{1\})$ may be exponentially long.

This, however, is an existential argument, and does not imply anything on the translation complexity. In particular, it is possible that, in order to get an exponential output complexity, the translation complexity has to increase as well, which weakens the argument. We now address this issue.

Observe that our translation of a formula $f(p_1, \ldots, p_n)$ to an equivalent Boolean formula with respect to some predicate $P \subseteq [0, 1]$ involves, in the worst case, evaluating f on every $n-$tuple of inputs it can take. Thus, the translation complexity is at least polynomial space for every set $\mathcal{F}$ of functions. If the functions in $\mathcal{F}$ are themselves computable in PSPACE, they are not the bottleneck of the translation, and the translation complexity remains in polynomial space. Thus, our goal is to find a Boolean function $f : \{0, 1\}^n \to \{0, 1\}$ that witnesses the exponential output complexity and is still computable in PSPACE.

The class NC^1 of Boolean functions for which there is a polynomial-size Boolean propositional formula is known to be a proper subset of PSPACE (e.g., see Chapter 6 in Arora and Barak [2009]); thus, we can choose the function f presented earlier to be PSPACE-complete, hence computable in PSPACE and not in NC^1.

We conclude that a super-polynomial[6] blowup in output complexity is unavoidable even if we restrict to functions for which the translation complexity is PSPACE. Formally, we have the following.

THEOREM 2.8. *There exists a set of functions $\mathcal{F}$ and* LTL[$\mathcal{F}$] *formulas φ_n for every $n \in \mathbb{N}$ such that the following hold:*

(*1*) *$Bool(\varphi_n, \{1\})$ is computable from φ_n in PSPACE.*
(*2*) *For every* LTL *formula ψ such that $\psi \equiv Bool(\varphi_n, \{1\})$, we have that $|\psi|$ is super-polynomial in $|\varphi_n|$.*

[6]In fact, since $\mathrm{NC} = \bigcup_{i \in \mathbb{N}} \mathrm{NC}^i$ is a proper subset of PSPACE, we can strengthen the claim to an output complexity blowup of $2^{\mathrm{polylog}(n)}$.

2.4. Translating LTL[$\mathcal{F}$] to Automata

The *automata-theoretic* approach uses the theory of automata as a unifying paradigm for system specification, verification, and synthesis [Thomas 1990; Vardi and Wolper 1994]. By viewing computations as words (over the alphabet of possible assignments to variables of the system), we can view both the system and its specification as languages. Decision problems about specifications and systems can then be reduced to problems about automata, like nonemptiness and containment. Automata-based methods have been implemented in both academic and industrial automated-verification tools (see FormalCheck and SPIN) [Kurshan 1998; Holzmann 2004]. In this section, we describe an automata-theoretic framework for reasoning about LTL[$\mathcal{F}$] specifications.

One approach is to develop a framework that is based on *weighted automata*. Like LTL[$\mathcal{F}$] formulas, weighted automata map words to values that are richer than `True` and `False` and, in particular, can map words to values in $[0, 1]$ [Mohri 1997; Droste et al. 2009]. Reasoning about weighted automata is, however, much more complex than reasoning about standard Boolean automata. Fundamental problems that have been solved decades ago for Boolean automata are still open or known to be undecidable in the weighted setting. This includes the problem of deciding whether a given nondeterministic weighted automaton can be determinized, and the problem of deciding whether the language of one automaton is contained (in the weighted sense) in the language of another automaton [Krob 1994]. Moreover, the semantics of weighted automata is typically simple (e.g., consists of operations on a semiring), whereas in our approach we need to apply arbitrary functions. Thus, even restricted forms of weighted automata, for which more problems are decidable (e.g., Kirsten and Lombardy [2009] and Filiot et al. [2014]), are not suitable. Accordingly, a second approach, which is the one that we follow, is to try and reduce the quantitative questions about LTL[$\mathcal{F}$] to questions about Boolean automata. The fact that LTL[$\mathcal{F}$] formulas have finitely many possible satisfaction values suggests that this is possible. The main challenge is to maintain the simplicity of the automata-theoretic framework of LTL in spite of the fact that the number of satisfaction values is exponential. In order to appreciate this challenge, note that applying the translation from Theorem 2.7 would result in algorithms that are exponentially more complex than those of LTL.

In order to explain our framework, let us recall first the translation of LTL formulas to *nondeterministic generalized Büchi automata* (NGBAs), as introduced in Vardi and Wolper [1986]. We start with the definition of NBAs and NGBAs. An NGBA is $\mathcal{A} = \langle \Sigma, Q, Q_0, \delta, \alpha \rangle$, where Σ is the input alphabet, Q is a finite set of states, $Q_0 \subseteq Q$ is a set of initial states, $\delta : Q \times \Sigma \to 2^Q$ is a transition function, and $\alpha \subseteq 2^Q$ is a set of sets of accepting states. The number of sets in α is the *index* of $\mathcal{A}$. When the index of an NGBA is 1 it is called a (standard) NBA. A run $r = r_0, r_1, \cdots$ of $\mathcal{A}$ on a word $w = w_1 \cdot w_2 \cdots \in \Sigma^\omega$ is an infinite sequence of states such that $r_0 \in Q_0$, and, for every $i \geq 0$, we have that $r_{i+1} \in \delta(r_i, w_{i+1})$. We denote by $\inf(r)$ the set of states that r visits infinitely often, that is, $\inf(r) = \{q : r_i = q \text{ for infinitely many } i \in \mathbb{N}\}$. The run r is *accepting* if it visits all the sets in α infinitely often. Formally, for every set $F \in \alpha$, we have that $\inf(r) \cap F \neq \emptyset$. An automaton accepts a word if it has an accepting run on it. The language of an automaton $\mathcal{A}$, denoted $L(\mathcal{A})$, is the set of words that $\mathcal{A}$ accepts.

In the Vardi-Wolper translation of LTL formulas to NGBAs [Vardi and Wolper 1986], each state of the automaton is associated with a set of formulas, and the NGBA accepts a computation from a state q if and only if the computation satisfies exactly all the formulas associated with q. The state space of the NGBA contains only states associated with maximal and consistent sets of formulas, the transitions are defined so that requirements imposed by temporal formulas are satisfied, and the acceptance condition is used in order to guarantee that requirements that involve the satisfaction of eventualities are not delayed forever.

In our construction here, each state of the NGBA assigns a satisfaction value to every subformula. Consistency then ensures that the satisfaction values agree with the functions in $\mathcal{F}$. Similar adjustments are made to the transitions and the acceptance condition. The construction translates an LTL[$\mathcal{F}$] formula φ to an NGBA, while setting its initial states according to a required predicate $P \subseteq [0,1]$. We then have that, for every computation $\pi \in (2^{AP})^\omega$, the resulting NGBA accepts π if and only if $[\![\pi, \varphi]\!] \in P$.

We note that a similar approach is taken in Faella et al. [2008], in which LTL formulas are interpreted over quantitative systems. The important difference is that the values in our construction arise from the formula and the functions that it involves, whereas in Faella et al. [2008], they are induced by the values of the atomic propositions.

Finally, we remark that, in the case that the input for our construction is a Boolean LTL formula, the constructed NGBA is almost identical to the one obtained from the Vardi-Wolper construction, with the only difference being that we do not start by pushing the negations in the formula to the atomic propositions (as this is generally not possible in an LTL[$\mathcal{F}$] formula).

THEOREM 2.9. *Let φ be an* LTL[$\mathcal{F}$] *formula and $P \subseteq [0,1]$ be a predicate. There exists an NGBA $\mathcal{A}_{\varphi,P}$ such that, for every computation $\pi \in (2^{AP})^\omega$, it holds that $[\![\pi, \varphi]\!] \in P$ if and onlyl if $\mathcal{A}_{\varphi,P}$ accepts π. Furthermore, $\mathcal{A}_{\varphi,P}$ has at most $2^{(|\varphi|^2)}$ states and its index is at most $|\varphi|$.*

PROOF. We define $\mathcal{A}_{\varphi,P} = \langle 2^{AP}, Q, \delta, Q_0, \alpha\rangle$ as follows. Let $cl(\varphi)$ be the set of φ's subformulas. Let C_φ be the collection of functions $g : cl(\varphi) \to [0,1]$ such that, for all $\psi \in cl(\varphi)$, we have that $g(\psi) \in V(\psi)$. Note that, by Lemma 2.5, the set $V(\psi)$ is finite for every formula ψ, thus so is C_φ. For a function $g \in C_\varphi$, we say that g is *consistent* if for every $\psi \in cl(\varphi)$, the following hold:

—If $\psi = \texttt{True}$, then $g(\psi) = 1$, and if $\psi = \texttt{False}$, then $g(\psi) = 0$.
—If $\psi = p \in AP$, then $g(\psi) \in \{0,1\}$.
—If $\psi = f(\psi_1, \ldots, \psi_k)$, then $g(\psi) = f(g(\psi_1), \ldots, g(\psi_k))$.

The state space Q of $\mathcal{A}_{\varphi,P}$ is the set of all consistent functions in C_φ. Then, $Q_0 = \{g \in Q : g(\varphi) \in P\}$ contains all states in which the value assigned to φ is in P.

We now define the transition function δ. For functions $g, g' \in Q$ and a letter $\sigma \in \Sigma$, we have that $g' \in \delta(g, \sigma)$ if and only if the following hold:

—$\sigma = \{p \in AP : g(p) = 1\}$.
—For all $\mathsf{X}\psi_1 \in cl(\varphi)$, we have that $g(\mathsf{X}\psi_1) = g'(\psi_1)$.
—For all $\psi_1 \mathsf{U} \psi_2 \in cl(\varphi)$, we have that $g(\psi_1 \mathsf{U} \psi_2) = \max\{g(\psi_2), \min\{g(\psi_1), g'(\psi_1 \mathsf{U} \psi_2)\}\}$.

Finally, every formula $\psi_1 \mathsf{U} \psi_2 \in cl(\varphi)$ contributes to α the set $F_{\psi_1 \mathsf{U} \psi_2} = \{g : g(\psi_2) = g(\psi_1 \mathsf{U} \psi_2)\}$.

We proceed to prove the correctness of the construction and to analyze the size of $\mathcal{A}_{\varphi,P}$. We first prove that if $\pi \in (2^{AP})^\omega$ is such that $[\![\pi, \varphi]\!] \in P$, then $\mathcal{A}_{\varphi,P}$ accepts π. For every $i \in \mathbb{N}$, consider the function $g_i \in C_\varphi$, where, for all $\psi \in cl(\varphi)$, we have that $g_i(\psi) = [\![\pi^i, \psi]\!]$. Clearly, g_i is consistent for all i. We claim that $r = g_0, g_1, \ldots$ is an accepting run of $\mathcal{A}_{\varphi,P}$ on π. First, $g_0 \in Q_0$ as $g_0(\varphi) \in P$. As for the transitions between states, it is easy to see that all the conditions between the transitions are satisfied. In particular, for formulas of the form $\psi_1 \mathsf{U} \psi_2$, assume that $[\![\pi^i, \psi_1 \mathsf{U} \psi_2]\!] = x$. By the semantics of U, one of the following holds:

—$[\![\pi^i, \psi_2]\!] = x$ and $\min\{[\![\pi^i, \psi_1]\!], [\![\pi^{i+1}, \psi_1 \mathsf{U} \psi_2]\!]\} \leq x$.
—$[\![\pi^i, \psi_2]\!] \leq x$ and $\min\{[\![\pi^i, \psi_1]\!], [\![\pi^{i+1}, \psi_1 \mathsf{U} \psi_2]\!]\} = x$.

Since this coincides with the condition for a transition between g_i and g_{i+1}, the transition is legal.

Finally, r visits every set in α infinitely often. Consider a subformula of the form $\psi_1 \mathsf{U} \psi_2$. If $[\![\pi^i, \psi_1 \mathsf{U} \psi_2]\!] = y$, then, by the semantics of U, there is an index $i \leq j$ such that $[\![\pi^j, \psi_1 \mathsf{U} \psi_2]\!] = [\![\pi^j, \psi_2]\!]$. Thus, the state g_j is in $F_{\psi_1 \mathsf{U} \psi_2}$. Thus, π is accepted by $\mathcal{A}_{\varphi,P}$.

The other direction is more complicated. Let $\pi \in (2^{AP})^\omega$ be such that π is accepted by $\mathcal{A}_{\varphi,P}$. We prove that $[\![\pi, \varphi]\!] \in P$. Let $\rho = g_1, g_2, \ldots$ be an accepting run of $\mathcal{A}_{\varphi,P}$ on π, and let $h_1, h_2, \ldots$ be a sequence of functions in C_φ such that, for all $i \in \mathbb{N}$ and $\psi \in cl(\varphi)$, we have that $h_i(\psi) = [\![\pi^i, \psi]\!]$. We claim that $h_i = g_i$ for all $i \in \mathbb{N}$. The proof is by induction on the structure of the formulas in $cl(\varphi)$. Consider a formula $\psi \in cl(\varphi)$.

—Let $\psi = \texttt{True}$ or $\psi = \texttt{False}$. By consistency, $h_i(\texttt{True}) = g_i(\texttt{True}) = 1$ and $h_i(\texttt{False}) = g_i(\texttt{False}) = 0$, and we are done.
—Let $\psi = p \in AP$. Since ρ is a legal run and transitions are labeled by the set of atomic propositions that are mapped to 1, we have that $h_i(p) = 1$ if and only if $[\![\pi^i, p]\!] = 1$ if and only if $p \in \pi_i$ if and only if $g_i(p) = 1$, and we are done.
—Let $\psi = f(\psi_1, \ldots, \psi_k)$. Since the state space of $\mathcal{A}_{\varphi,P}$ contains only consistent functions, the claim follows from the induction hypothesis.
—Let $\psi = \mathsf{X}\psi_1$. Since ρ is a legal run and transitions follow the semantics of X, the claim follows from the induction hypothesis.
—Let $\psi = \psi_1 \mathsf{U} \psi_2$. In Lemma 2.10, we prove that since ρ is an accepting run, then

$$g_i(\psi_1 \mathsf{U} \psi_2) = \max_{i \leq j} \left\{ \min \left\{ g_j(\psi_2), \min_{i \leq k < j} \{ g_k(\psi_1) \} \right\} \right\}.$$

By the induction hypothesis, this expression equals

$$\max_{i \leq j} \left\{ \min \left\{ [\![\pi^j, \psi_2]\!], \min_{i \leq k < j} \left\{ [\![\pi^k, \psi_1]\!] \right\} \right\} \right\},$$

which, by the semantics of LTL[$\mathcal{F}$], is exactly $[\![\pi^i, \psi]\!] = h_i(\psi)$.

We conclude that $h_1 = g_1$. Since g_1 is an initial state, it follows that $[\![\pi, \varphi]\!] \in P$, and we are done.

It is left to prove that the number of states in $\mathcal{A}_{\varphi,P}$ is at most $2^{|\varphi|^2}$. Let $n = |\varphi|$. Recall that $V(\varphi)$ is the set of possible satisfaction values of φ. Let $V^+(\varphi) = \bigcup_{\psi \in cl(\varphi)} V(\psi)$. Thus, $V^+(\varphi)$ includes also the satisfaction values of the subformulas of φ. Clearly, the set of functions C_φ, and hence the set of states Q, is contained in the set of functions $g : cl(\varphi) \to V^+(\varphi)$. We first claim that $|V^+(\varphi)| \leq 2^n$. The proof is similar to the proof of Lemma 2.5, and proceeds by induction on the structure of φ. The induction steps are similar to the steps there, with the only nontrivial change being the case $\varphi = f(\psi_1, \ldots, \psi_k)$. Here, $|V^+(\varphi)| = |V(\varphi)| + \sum_{i=1}^k |V^+(\psi_i)|$. By the induction hypothesis, we have that $|V^+(\psi_i)| \leq 2^{|\psi_i|}$, implying that $|V^+(\varphi)| \leq 2 \cdot 2^{\sum_{i=1}^k |\psi_i|} = 2^n$; thus, we are done.

Now, since $|cl(\varphi)| \leq n$ and $|V^+(\varphi)| \leq 2^n$, it follows that Q, which is contained in $(V^+(\varphi))^{cl(\varphi)}$, is of size at most 2^{n^2}. Finally, the index of $\mathcal{A}_{\varphi,P}$ is bounded by the number of subformulas of the form $\psi_1 \mathsf{U} \psi_2$, hence bounded by n.

LEMMA 2.10. *Using the notations in the proof of Theorem 2.9, we have that*

$$g_i(\psi_1 \mathsf{U} \psi_2) = \max_{i \leq j} \left\{ \min \left\{ g_j(\psi_2), \min_{i \leq k < j} \{ g_k(\psi_1) \} \right\} \right\}.$$

PROOF. Since ρ is a legal run, then, for every $i \in \mathbb{N}$, it holds that

$$g_i(\psi_1 \mathsf{U} \psi_2) = \max \{ g_i(\psi_2), \min \{ g_i(\psi_1), g_{i+1}(\psi_1 \mathsf{U} \psi_2) \} \}. \quad (*)$$

We prove the lemma by proving two inequalities. We start by proving that

$$g_i(\psi_1 \mathsf{U} \psi_2) \leq \max_{i \leq j} \left\{ \min \left\{ g_j(\psi_2), \min_{i \leq k < j} \{ g_k(\psi_1) \} \right\} \right\}.$$

Let g_i be a state in ρ. Since ρ is accepting, there exists an index l such that $i \leq l$ and $g_l \in F_{\psi_1 \mathsf{U} \psi_2}$. Let l be the minimal such index. We prove the inequality by induction on $l - i$. If $l = i$, then, as $g_l \in F_{\psi_1 \mathsf{U} \psi_2}$, we have that $g_i(\psi_2) = g_i(\psi_1 \mathsf{U} \psi_2)$. Since $g_i(\psi_2)$ is the first element in the max at hand, we conclude the inequality for the base case. Assume correctness for $l - (i + 1)$; we prove correctness for $l - i$. Since g_{i+1} succeeds g_i in ρ, we have that $g_i(\psi_1 \mathsf{U} \psi_2) = \max \{ g_i(\psi_2), \min \{ g_i(\psi_1), g_{i+1}(\psi_1 \mathsf{U} \psi_2) \} \}$. Plugging the induction hypothesis for g_{i+1}, we get that

$$g_i(\psi_1 \mathsf{U} \psi_2) \leq \max \left\{ g_i(\psi_2), \min \left\{ g_i(\psi_1), \max_{i+1 \leq j} \left\{ \min \left\{ g_j(\psi_2), \min_{i+1 \leq k < j} \{ g_k(\psi_1) \} \right\} \right\} \right\} \right\}.$$

It is a simple exercise to verify that the latter equals

$$\max_{i \leq j} \left\{ \min \left\{ g_j(\psi_2), \min_{i \leq k < j} \{ g_k(\psi_1) \} \right\} \right\},$$

and we are done.

For the second direction, we proceed similarly. Let l be the minimal index $i \leq l$ such that g_l is accepting. The proof is by induction over $l - i$. If $l = i$, we have that $g_i(\psi_2) = g_i(\psi_1 \mathsf{U} \psi_2)$ (since g_i is accepting). We claim that

$$g_i(\psi_2) = g_i(\psi_1 \mathsf{U} \psi_2) = \max_{i \leq j} \left\{ \min \left\{ g_j(\psi_2), \min_{i \leq k < j} \{ g_k(\psi_1) \} \right\} \right\}.$$

Assume by way of contradiction that this does not hold. Thus, there exists some $k > i$ such that $g_i(\psi_2) = g_i(\psi_1 \mathsf{U} \psi_2) < \min \{ g_k(\psi_2), \min_{i \leq j < k} \{ g_j(\psi_1) \} \}$. From $(*)$, we make two observations. First, for every state g from which there exists an outgoing edge, we have that $g(\psi_1 \mathsf{U} \psi_2) \geq g(\psi_2)$ (otherwise the max is not met). Second, assume that there is a transition from g to g' and that $g(\psi_1 \mathsf{U} \psi_2) < g'(\psi_1 \mathsf{U} \psi_2)$; then, we have that $g(\psi_1 \mathsf{U} \psi_2) \geq \max \{ g(\psi_1), g(\psi_2) \}$ (this follows from considering the different ordering of the elements of $(*)$). We call this the *stepping-up* rule, with the intuition being that, in order to increase $g(\psi_1 \mathsf{U} \psi_2)$, one must "step on" (i.e., be greater than) $g(\psi_1)$ and $g(\psi_2)$. Since $g_i(\psi_1) > g_i(\psi_1 \mathsf{U} \psi_2)$, then, from the negation of the stepping-up rule, it follows that $g_{i+1}(\psi_1 \mathsf{U} \psi_2) \leq g_i(\psi_1 \mathsf{U} \psi_2)$. Applying the same logic inductively, using the assumption that $\min \{ g_i(\psi_1), \ldots, g_{k-1}(\psi_1), g_k(\psi_2) \}$ is greater than $g_i(\psi_1 \mathsf{U} \psi_2)$, we get that $g_i(\psi_1 \mathsf{U} \psi_2) \geq g_k(\psi_1 \mathsf{U} \psi_2) \geq g_k(\psi_2) > g_i(\psi_1 \mathsf{U} \psi_2)$, which is a contradiction (this is not trivial, but it is a simple technical exercise).

Finally, we complete the inductive step exactly as the previous case, by plugging in $\leq$ instead of $\geq$ in the induction assumption. □

Remark 2.11. Observe that while the size of $\mathcal{A}_{\varphi,P}$ described in Theorem 2.9 is at most $2^{(|\varphi|^2)}$, in order to compute $\mathcal{A}_{\varphi,P}$, we must be able to evaluate the functions in $\mathcal{F}$ as well as test membership in P. Specifically, traversing the successors of a state g in $\mathcal{A}_{\varphi,P}$ can be done in PSPACE, provided that evaluating the functions in $\mathcal{F}$, and that testing membership in P, can be done in PSPACE. If these assumptions do not hold, then computing the functions in $\mathcal{F}$ or testing membership in P becomes the computational bottleneck of the computation, as was the case in Section 2.3.1.

Remark 2.12. The construction described in the proof of Theorem 2.9 is such that selecting the set of initial states allows us to specify any (propositional) condition regarding the subformulas of φ. A simple extension of this idea allows us to consider

a set of formulas $\{\varphi_1, \ldots, \varphi_m\} = \Phi$ and a predicate $P \subseteq [0,1]^m$, as well as to construct an NGBA that accepts a computation π if and only if $\langle [\![\pi, \varphi_1]\!], \ldots, [\![\pi, \varphi_n]\!] \rangle \in P$. The state space of the product consists of functions that map all the formulas in Φ to their satisfaction values; we only have to choose as the initial states these functions g for which $\langle g(\varphi_1), \ldots, g(\varphi_n) \rangle \in P$. As we shall see in Section 2.5, this also allows us to use the automata-theoretic approach in order to examine relations between the satisfaction values of different formulas.

2.5. Solving the Questions for LTL[$\mathcal{F}$]

In this section, we solve the search questions defined in Section 2.2. We show that they all can be solved for LTL[$\mathcal{F}$] with roughly the same complexity as for LTL. When we analyze complexity, we assume that the functions in $\mathcal{F}$ can be computed in a complexity that is subsumed by the complexity of the problem for LTL (PSPACE, except for 2EXPTIME for realizability), which is very reasonable. Otherwise, computing the functions becomes the computational bottleneck (see Section 2.3.1).

2.5.1. Solving the Search Questions. The verification and synthesis questions in the quantitative setting are basically search problems, asking for the best or worst value (see Section 2.2). Since every LTL[$\mathcal{F}$] formula may only have exponentially many satisfaction values, one can reduce a search problem to a set of decision problems with respect to specific thresholds, remaining in PSPACE. Combining this with the construction of NGBAs described in Theorem 2.9 is the key to our algorithms.

We can now describe the algorithms in detail.

—**Satisfiability and validity.** We start with satisfiability and solve the decision version of the problem: given φ and a threshold $v \in [0,1] \cap \mathbb{Q}$, decide whether there exists a computation π such that $[\![\pi, \varphi]\!] \geq v$. The latter can be solved by checking the nonemptiness of the NGBA $\mathcal{A}_{\varphi,P}$ with $P = [v, 1]$. Since the NGBA can be constructed on-the-fly (i.e., constructing the states as needed during the execution of the algorithm, without keeping the entire structure in memory), this can be done in PSPACE in the size of $|\varphi|$. The search version can be solved in PSPACE by iterating over the set of relevant thresholds.

We proceed to validity. It is not hard to see that, for all φ and v, we have that $\forall \pi, [\![\pi, \varphi]\!] \geq v$ if and only if $\neg(\exists \pi,\ [\![\pi, \varphi]\!] < v)$. The latter can be solved by checking, in PSPACE, the nonemptiness of the NGBA $\mathcal{A}_{\varphi,P}$ with $P = [0, v)$. Since PSPACE is closed under complementation, we are done. In both cases, the nonemptiness algorithm can return a witness, when it exists.

—**Implication and equivalence.** In the Boolean setting, implication can be reduced to validity, which is, in turn, reduced to satisfiability. Doing the same here is more sophisticated, but possible: we add to the given set $\mathcal{F}$ the average and negation operators. It is not hard to verify that, for every computation π, it holds that $[\![\pi, \varphi_1 \oplus_{\frac{1}{2}} \neg\varphi_2]\!] = \frac{1}{2}[\![\pi, \varphi_1]\!] + \frac{1}{2}(1 - [\![\pi, \varphi_2]\!]) = \frac{1}{2}([\![\pi, \varphi_1]\!] - [\![\pi, \varphi_2]\!]) + \frac{1}{2}$. In particular, $\max\{[\![\pi, \varphi_1]\!] - [\![\pi, \varphi_2]\!] : \pi \text{ is a computation}\} = 2 \cdot \max\{[\![\pi, \varphi_1 \oplus_{\frac{1}{2}} \neg\varphi_2]\!] : \pi \text{ is a computation}\} - 1$. Thus, the problem reduces to the satisfiability of $\varphi_1 \oplus_{\frac{1}{2}} \neg\varphi_2$, which is solvable in PSPACE. Note that, alternatively, one can proceed as suggested in Remark 2.12 and reason about the composition of the NGBAs for φ_1 and φ_2. The solution to the equivalence problem is similar, by checking both directions of the implication.

—**Model checking.** The complement of the problem, namely, whether there exists a computation π of $\mathcal{K}$ such that $[\![\pi, \varphi]\!] < v$, can be solved by taking the product of the NGBA $\mathcal{A}_{\varphi,(0,v]}$ from Theorem 2.9 with the system $\mathcal{K}$ and checking for emptiness on-the-fly. As in the Boolean case, this can be done in PSPACE. Moreover, in the case that the product is not empty, the algorithm returns a witness: a computation of $\mathcal{K}$

that satisfies φ with a low quality. We note that in the case of a single computation, motivated by multivalued monitoring [Donzé et al. 2012], one can label the computation in a bottom-up manner, as in CTL model checking, and the problem can be solved in polynomial time.

—**Realizability and synthesis.** Several algorithms are suggested in the literature for solving the LTL realizability problem [Pnueli and Rosner 1989]. Since they are all based on a translation of specifications to automata, we can adopt them. Here, we describe an adoption of the Safraless algorithm of Kupferman and Vardi [2005] and its extension to NGBAs [Kupferman et al. 2006]. Given φ and v, the algorithm starts by constructing the NGBA $\mathcal{A}_{\varphi,[0,v)}$ and dualizing it to a universal generalized co-Büchi automaton (UGCW) $\tilde{\mathcal{A}}_{\varphi,[0,v)}$. Since dualization amounts to complementation, $\tilde{\mathcal{A}}_{\varphi,[0,v)}$ accepts exactly all computations π with $[\![\pi, \varphi]\!] \geq v$. Being universal, we can expand $\tilde{\mathcal{A}}_{\varphi,[0,v)}$ to a universal tree automaton $\mathcal{U}$ that accepts a tree with directions in 2^I and labels in 2^O if all its branches, which correspond to input sequences, are labeled by output sequences such that the composition of the input and output sequences is a computation accepted by $\tilde{\mathcal{A}}_{\varphi,[0,v)}$. Realizability then amounts to checking the nonemptiness of $\mathcal{U}$ and synthesis to finding a witness to its nonemptiness. Since φ only has an exponential number of satisfaction values, we can solve the realizability and synthesis search problems by repeating this procedure for all relevant values. Since the size of $\mathcal{A}_{\varphi,[0,v)}$ is singly exponential in $|\varphi|$, the complexity is the same as in the Boolean case, namely, 2EXPTIME-complete.

Remark 2.13. Recall that a solution to search and optimization problems that correspond to the decision problems outlined earlier involves the solution of multiple decision problems. Observe that, for all the problems described earlier, the solution to the multiple decision problems involve reasoning on NGBAs that differ only in their initial states. It is thus possible to combine different searches. The drawback of this "simultaneous search approach" is that we cannot use the result of earlier searches in order to direct the search; thus, we are no longer in PSPACE. Several approaches, however, for checking the nonemptiness prefer to give up on-the-flyness and reason about the automata in a global way. Such approaches can solve the decision problem with respect to all values in $V(\varphi)$ simultaneously.

3. EXTENSIONS AND RESTRICTIONS OF PROPOSITIONAL QUALITY

The clean translation of LTL[$\mathcal{F}$] to known Boolean frameworks (e.g., Boolean automata and LTL) suggests that extending LTL[$\mathcal{F}$] to richer structures (e.g., weighted systems) or other specification formalism (e.g., branching temporal logics) may be possible. In Sections 3.1 to 3.3, we study such extensions. Then, in Section 3.4, we study the fragment LTL$^{\triangledown}$ of LTL[$\mathcal{F}$]. Beyond a simpler automata-theoretic approach, the importance of LTL$^{\triangledown}$ would become apparent in Section 6.1, in which we consider the combination of LTL[$\mathcal{F}$] with discounting temporal operators. As we show there, while such a combination results in an undecidable logic, it is possible to retain decidability when discounting temporal operators are combined with the fragment LTL$^{\triangledown}$.

3.1. Extending LTL[$\mathcal{F}$] to Weighted Systems

A *weighted Kripke structure* is a tuple $\mathcal{K} = \langle AP, S, I, \rho, L\rangle$, where AP, S, I, and ρ are as in Boolean Kripke structures, and $L : S \to [0, 1]^{AP}$ maps each state to a weighted assignment to the atomic propositions. Thus, the value $L(s)(p)$ of an atomic proposition $p \in AP$ in a state $s \in S$ is a value in $[0, 1]$. The semantics of LTL[$\mathcal{F}$] with respect to a weighted computation coincides with the one for nonweighted systems, except that, for an atomic proposition p, we have that $[\![\pi, p]\!] = L(\pi_0)(p)$.

It is not hard to extend the construction of $\mathcal{A}_{\varphi,P}$, as described in the proof of Theorem 2.9, to the case of weighted systems, as we now describe.

Assume that there is a finite set $\Upsilon \subseteq [0,1]^{AP}$ such that the values of the weighted atomic propositions are in Υ. In particular, in the model-checking problem, we have that Υ is such that $c \in \Upsilon$ if and only if there exists a state $s \in S$ such that $L(s) = c$. The construction of $\mathcal{A}_{\varphi,P}$ is then modified as follows. We adjust the transitions so that there is a transition from state g with letter $\sigma \in \Upsilon$ only if g agrees with σ on the values of the atomic propositions. Hence, in settings in which the values for the atomic propositions are known—in particular, model checking—the solutions to the search questions are similar to the ones described for LTL[$\mathcal{F}$] with Boolean atomic propositions. Formally, we have the following theorem.

THEOREM 3.1. *Let $\mathcal{K}$ be a weighted Kripke structure, φ be an* LTL[$\mathcal{F}$] *formula, and $P \subseteq [0,1]$ be a predicate. There exists an NGBA $\mathcal{A}_{\mathcal{K},\varphi,P}$ such that, for every computation π of $\mathcal{K}$, it holds that $[\![\pi, \varphi]\!] \in P$ if and only if $\mathcal{A}_{\mathcal{K},\varphi,P}$ accepts π.*

We remark that, in the weighted-systems setting, the satisfiability (resp., synthesis) problem, namely, the problem of finding a weighted path (resp., weighted system) that maximizes the satisfaction value of a given formula, remains open.

3.2. LTL[$\mathcal{F}$] over Finite Computations

In the Boolean setting, one can interpret LTL over finite computations [Manna and Pnueli 1995]. The main change in the semantics is the evaluation of the X operator in the last position in the computation, in which $\mathsf{X}\varphi$ can be arbitrarily defined as `True` or as `False`.

Similarly, one can interpret LTL[$\mathcal{F}$] over finite computations. Consider a computation $\pi = \pi_0, \pi_1, \cdots \pi_n \in (2^{AP})^*$. The semantics is defined similarly to the semantics for infinite computations described in Table I, with two exceptions. First, $[\![\pi, \varphi_1 \mathsf{U} \varphi_2]\!] = \max_{0 \le i \le n}\{\min\{[\![\pi^i, \varphi_2]\!], \min_{0 \le j < i}[\![\pi^j, \varphi_1]\!]\}\}$. Second, $[\![\pi, \mathsf{X}\varphi]\!]$ is $[\![\pi^1, \varphi]\!]$ when $n \ge 1$ (i.e., when the computation is of length at least 2), and is 0 when $n = 0$ (i.e., when the computation is of length 1). Note that the formula $\mathsf{X}\varphi$ evaluates to 0 in the last position in the computation for every formula φ, which is known as the *strong* semantics. One could also define the X operator to evaluate to 1 in the last position, known as the *weak* semantics. It is not hard to see that the two semantics have the same expressive power.

We solve the basic questions for LTL[$\mathcal{F}$] over finite computations via the same approach that we took in Section 2.4, by translating an LTL[$\mathcal{F}$] formula to an automaton. The construction is similar to the one presented in Theorem 2.9, with the difference being that we construct an NFA, rather than an NGBA. Specifically, given an LTL[$\mathcal{F}$] formula φ and a predicate $P \subseteq [0,1]$, we construct the NFA $\mathcal{A}_{\varphi,P} = \langle 2^{AP}, Q, \delta, Q_0, \alpha \rangle$ where Q, δ and Q_0 are as defined in the proof of Theorem 2.9. Recall that a state of Q is a function $g : cl(\varphi) \to [0,1]$. Then, the acceptance condition $\alpha \subseteq Q$ consists of all states $g \in Q$ such that the following hold:

—For all $\psi_1 \mathsf{U} \psi_2 \in cl(\varphi)$, we have that $g(\psi_2) = g(\psi_1 \mathsf{U} \psi_2)$.
—For all $\mathsf{X}\psi \in cl(\varphi)$, we have that $g(\mathsf{X}\psi) = 0$.

The correctness of the construction can be proved similarly to Theorem 2.9.

3.3. Formalizing Quality with Branching Temporal Logics

Formulas of LTL[$\mathcal{F}$] specify ongoing behaviors of linear computations. A Kripke structure is not linear, and the way we interpret LTL[$\mathcal{F}$] formulas with respect to it is universal. In *branching temporal logic*, one can add universal and existential quantifiers to the syntax of the logic, and specifications can refer to the branching nature of the system [Emerson and Halpern 1986].

We define the branching temporal logic CTL$^\star[\mathcal{F}]$ to extend the Boolean branching temporal logic CTL* by propositional quality operators; Equivalently, CTL$^\star[\mathcal{F}]$ is defined as the extension of LTL$[\mathcal{F}]$ with the path quantifiers E and A. Formulas of the form $\mathsf{E}\varphi$ and $\mathsf{A}\varphi$ are referred to as *state formulas*; they are interpreted over states s in the structure with the semantics $[\![s, \mathsf{E}\varphi]\!] = \max\{[\![\pi, \varphi]\!] : \pi \text{ is a path that starts in } s\}$ and $[\![s, \mathsf{A}\varphi]\!] = \min\{[\![\pi, \varphi]\!] : \pi \text{ is a path that starts in } s\}$.

Emerson and Lei [1985] describe a general technique for extending the scope of LTL model-checking algorithms to CTL*. The idea is to repeatedly consider an innermost state subformula, view it as an (existentially or universally quantified) LTL formula, apply LTL model checking in order to evaluate it in all states, and add a fresh atomic proposition that replaces this subformula and holds in exactly these states that satisfy it.

A naïve attempt to use this technique in order to model check CTL$^\star[\mathcal{F}]$ fails, as the satisfaction value of LTL$[\mathcal{F}]$ formulas is not Boolean; hence, they cannot be replaced by atomic propositions. Fortunately, having solved the LTL$[\mathcal{F}]$ model-checking problem for weighted systems (see Section 3.1), we can still use a variant of this technique: rather than replacing formulas by Boolean atomic propositions, replace them by weighted ones, in which the value of a fresh weighted atomic proposition is found by solving the search variant of the LTL$[\mathcal{F}]$ model-checking problem. Hence, the model-checking problem for CTL$^\star[\mathcal{F}]$ is PSPACE-complete, as are the ones for LTL$[\mathcal{F}]$ and for CTL*. A direction for future research is to study fragments of CTL$^\star[\mathcal{F}]$ for which model checking is simpler. In particular, our initial results show that, for CTL$[\mathcal{F}]$, in which each temporal operator must be preceded by a path quantifier, it is possible to apply a variant of the linear-time fixed-point-based model-checking algorithm of CTL [Emerson and Lei 1986]. Thus, the model-checking problem for CTL$[\mathcal{F}]$ is in P.

More challenging is the handling of the other search problems. There, the solution involves a translation of CTL$^\star[\mathcal{F}]$ formulas to tree automata. Since the automata-theoretic approach for CTL* satisfiability and synthesis has the Vardi-Wolper construction at its heart [Kupferman et al. 2000], this is possible. We now elaborate on the solution.

Consider a CTL$^\star[\mathcal{F}]$ formula φ and a threshold v. We construct from φ an alternating automaton $\mathcal{A}_{\varphi,v}$ over trees, such that $\mathcal{A}_{\varphi,v}$ accepts a computation tree τ if and only if $[\![\tau, \varphi]\!] \geq v$. The construction is similar to that in Kupferman et al. [2000]. There, the construction uses the Vardi-Wolper construction to NBAs for inner (path) formulas, and combines these automata using alternations to obtain $\mathcal{A}_{\varphi,v}$. Then, the synthesis problem reduces to the emptiness problem for alternating automata, which can be solved in EXPTIME.

To adapt this construction to the case of CTL$^\star[\mathcal{F}]$, we only need to observe that the inner NBAs can be obtained with the construction in Theorem 2.9. Moreover, the size of the obtained NBAs is single exponential in φ.

Since the size of $\mathcal{A}_{\varphi,v}$ is exponential in the size of φ, we end up with a 2EXPTIME algorithm for the synthesis problem, matching the complexity of CTL* synthesis in the Boolean case, for which a matching 2-EXPTIME lower bound is known [Kupferman and Vardi 1997]. We conclude with the following theorem.

THEOREM 3.2. *The model-checking and synthesis problems for* CTL$^\star[\mathcal{F}]$ *are PSPACE-complete and 2EXPTIME complete, respectively.*

3.4. The Fragment LTL$^\triangledown$

We define here a fragment of LTL$[\mathcal{F}]$ that covers the possible linear approaches of formalizing quality as a value between true and false. Formally, we define the logic LTL$^\triangledown$ as a special case of LTL$[\mathcal{F}]$, for which the set of functions $\mathcal{F}$ contains the standard

Boolean operators and the unary operators $\triangledown_\lambda$, $\blacktriangledown_\lambda$, and $\blacktriangledown\!\!\!\!-_\lambda$, defined as follows. Let $x \in [0,1]$. Then,

$$\bullet\ \triangledown_\lambda(x) = \lambda \cdot x \qquad \bullet\ \blacktriangledown_\lambda(x) = \lambda x + (1-\lambda) \qquad \bullet\ \blacktriangledown\!\!\!\!-_\lambda(x) = \lambda \cdot x + (1-\lambda)/2$$

The three operators $\triangledown_\lambda$, $\blacktriangledown_\lambda$, and $\blacktriangledown\!\!\!\!-_\lambda$ are designed to capture the *competence*, *necessity*, and *confidence* of the specifications, as elaborated here.

The semantics of the operator $\triangledown_\lambda$ can be thought of as a measure of *competence*: the formula $\triangledown_{\frac{3}{4}}\varphi$ has the property that, even if φ is fully satisfied, it is still not competent enough to increase the satisfaction value beyond $\frac{3}{4}$. This semantics makes an implicit assumption that `True` corresponds to "happiness" and `False` to the *lack* of happiness.

However, it may be the case that we regard `True` as contentedness ("no problems"), and `False` as "critical problems." To capture this, we could have defined a different semantics, denoted $[\![\cdot,\cdot]\!]_{[-1,0]}$, whereby $[\![\pi, \texttt{True}]\!]_{[-1,0]} = 0$ and $[\![\pi, \texttt{False}]\!]_{[-1,0]} = -1$. With this view, we have the operator $\blacktriangledown_\lambda$ with the semantics $[\![\pi, \blacktriangledown_\lambda\varphi]\!]_{[-1,0]} = \lambda[\![\pi,\varphi]\!]_{[-1,0]}$. The operator $\blacktriangledown_\lambda$ can be thought of as a measure of *necessity*: the closer λ is to 1, the more necessary it is that φ gets a satisfaction value close to `True`. It should be noted that, in this semantics, we have $[\![\pi, \neg\varphi]\!]_{[-1,0]} = -(1 + [\![\pi,\varphi]\!]_{[-1,0]})$. It is easy to check that, in the framework of LTL[$\mathcal{F}$], whereby $[\![\pi, \texttt{True}]\!] = 1$ and $[\![\pi, \texttt{False}]\!] = 0$, this operator is translated to $\blacktriangledown_\lambda(x) = \lambda x + (1-\lambda)$.

The two operators $\triangledown$ and $\blacktriangledown$ are asymmetric with respect to `True` and `False`. A third, symmetric option is to consider a semantics, denoted $[\![\cdot,\cdot]\!]_{[-1,1]}$, whereby $[\![\pi, \texttt{True}]\!]_{[-1,1]} = 1$ and $[\![\pi, \texttt{False}]\!]_{[-1,1]} = -1$. This captures the intuition that `False` is the opposite of `True` (rather than the lack of `True`). In this semantics, $[\![\pi, \neg\varphi]\!]_{[-1,1]} = -[\![\pi,\varphi]\!]_{[-1,1]}$ and the operator $[\![\pi, \blacktriangledown\!\!\!\!-_\lambda\varphi]\!]_{[-1,1]} = \lambda[\![\pi,\varphi]\!]$. We use $\blacktriangledown\!\!\!\!-$ to describe *confidence*: `True` corresponds to full confidence that a formula holds, whereas `False` is full confidence that the formula does not hold. In the framework of LTL[$\mathcal{F}$], whereby $[\![\pi, \texttt{True}]\!] = 1$ and $[\![\pi, \texttt{False}]\!] = 0$, we have that $\blacktriangledown\!\!\!\!-_\lambda(x) = \lambda \cdot x + (1-\lambda)/2$.

The competence and the necessity operators are dual in the sense that, while the competence operator reduces the truth level, the necessity operator reduces the falseness level. Then, the confidence operator reduces both the truth and falseness levels. Together, the three operators cover the possible linear approaches of formalizing quality as a value between true and false.

The fragment LTL$^\triangledown$ has several appealing properties. Algorithmically, certain procedures are simpler for it. In particular, its decision problems can be polynomially translated to questions about LTL:

Observation 3.3. For LTL$^\triangledown$ formulas and for predicates of the form $[c,1]$, the translation to an equivalent LTL formula, as described in Theorem 2.7, is polynomial. Indeed, for such predicates we have that $Bool(\varphi_1 \mathsf{U} \varphi_2, [c,1]) = Bool(\varphi_1, [c,1]) \mathsf{U} Bool(\varphi_2, [c,1])$, and $Bool(\triangledown_\lambda\varphi, [c,1]) = Bool(\varphi, [\frac{c}{\lambda}, 1])$ if $\frac{c}{\lambda} \leq 1$ and `False` otherwise. The translation of $\blacktriangledown$ and $\blacktriangledown\!\!\!\!-$ is similar. Thus, no exponential blowup is involved in the translation, which means that we can solve the decision problems for LTL$^\triangledown$ by first converting its formulas to LTL formulas with no additional computational cost.

As a specification formalism, it provides ways to rank subformulas according to their necessity, competence or the confidence that we have in them. We demonstrate the usefulness of this fragment in the following examples.

Example 3.4. Consider the specification $\mathsf{G}(req \rightarrow grant \vee \mathsf{X}grant)$, and suppose that we are happier if the request is granted immediately and less happy if it is done in the next step (this resembles the example in Section 1.1.1, without the requirement that

the grant holds for two steps). This can be captured in LTL$^{\triangledown}$ using the formula

$$\mathsf{G}(req \rightarrow (grant \vee \triangledown_{\frac{3}{4}} \mathsf{X} grant)).$$

In this formula, we are fully happy only when requests are immediately satisfied. Postponing the satisfaction results in a small penalty, which is set by lowering the competence of $\mathsf{X}grant$ to $\frac{3}{4}$.

Next, assume that another desirable property is that we use grants sparingly. In particular, we do not want two grants to be given to a single request. This can be formulated by changing the earlier formula to

$$\mathsf{G}(req \rightarrow ((grant \wedge \neg \mathsf{X} grant) \vee \blacktriangledown_{\frac{3}{4}} (\neg grant \wedge \mathsf{X} grant) \vee \blacktriangledown_{\frac{1}{2}} (grant \wedge \mathsf{X} grant))).$$

Note that, here, $(grant \wedge \mathsf{X}grant)$ gives satisfaction value of $\frac{3}{4}$ (according to the semantics of $\blacktriangledown$ in the $[0, 1]$ range). Moreover, if there is a request that gets no grants, the satisfaction value is still $\frac{1}{4}$, representing the fact that, even though we failed to grant the specification, we at least managed to use grants sparingly.

In the following examples, we demonstrate how LTL$^{\triangledown}$ can be used for vacuity testing and coverage, as well as prioritizing safety requirements.

Example 3.5 (*Vacuity and Coverage*). Vacuity and coverage are two popular *sanity checks* in formal verification. They are applied after a successful verification process, aiming to check that all the components of the specification and the system have played a role in the satisfaction. Algorithms for measuring vacuity and coverage are based on model-checking mutations of the system or the specification [Kupferman 2006]. For example, after a system is proven to satisfy the specification $\mathsf{G}(req \rightarrow \mathsf{F}grant)$, vacuity-detection algorithms would mutate the specification to $\mathsf{G}(req \rightarrow \texttt{False})$ or to $\mathsf{G}(\texttt{True} \rightarrow \mathsf{F}grant)$ in order to check that all components of the specification affect the satisfaction. Coverage-measuring algorithms would then mutate the system, say, by removing some grants, in order to check whether all components of the system affect the satisfaction. A serious drawback of vacuity and coverage lies in the fact that it is difficult to distinguish between different cases of vacuous satisfaction and low coverage. While some cases should alarm the designer, some are a natural part of the design. In the earlier example, it is more alarming to discover that the specification has been satisfied in a system in which no requests are issued than to discover that the specification has been satisfied in a system in which infinitely many grants are issued. Using LTL$^{\triangledown}$, the designer can model vacuity and coverage in a way that would indicate the "level of alarm."

For example, keeping in mind that implications are disjunctions, we can refine the specification $\mathsf{G}(req \rightarrow \mathsf{F}grant)$ to $\mathsf{G}((\triangledown_{\frac{1}{3}} \neg req) \vee \mathsf{F}grant)$, which would get a satisfaction value $\frac{1}{3}$ in computations in which it is satisfied only thanks to the fact that there are only finitely many requests.

Example 3.6 (*Safety*). Consider a vending machine that serves drinks. A natural safety property is to require the machine to always have drinks, say, coffee and tea. Or should we say "coffee or tea"? The quality of a safety property is closely related to the necessity operator. The operator allows us to value the level of violating the safety requirements. For example, we may require that always having some drinks is a must, while having all drinks is a nice to have, using the formula $\mathsf{G}((coffee \vee tea) \wedge \blacktriangledown_{\frac{3}{4}} (coffee \wedge tea))$. This example can obviously be generalized to provide the necessity level of each subset of drinks. Another safety requirement for the machine is to always have coins for change. Since this is not a critical requirement, we can formalize it by $\blacktriangledown_{\frac{1}{8}} \mathsf{G} coins$.

4. FORMALIZING TEMPORAL QUALITY

In this section, we introduce the temporal logic $\text{LTL}^{\text{disc}}[\mathcal{D}]$ (Section 4.1) that provides formal means for specifying temporal quality. As has been the case of propositional quality, our algorithms for reasoning about $\text{LTL}^{\text{disc}}[\mathcal{D}]$ are based on a translation of a given $\text{LTL}^{\text{disc}}[\mathcal{D}]$ formula into an automaton (Section 4.2). The translation here, however, is much more involved, as a given formula might have infinitely many satisfaction values.

Using the automata-theoretic approach, we solve the decision problems for $\text{LTL}^{\text{disc}}[\mathcal{D}]$ (Section 4.3). The fact that a formula may have infinitely many satisfaction values not only makes the algorithms more complicated, but also means that a solution to the corresponding search and optimization problems does not follow, and we leave them open.

The complexity of solving the decision problems depends on the discounting functions in $\mathcal{D}$. For the set of exponential-discounting functions E, we analyze the concrete complexities and show that they stay in the same complexity classes of standard LTL (Section 4.4).

4.1. The Logic $\text{LTL}^{\text{disc}}[\mathcal{D}]$

The linear temporal logic $\text{LTL}^{\text{disc}}[\mathcal{D}]$ generalizes LTL by adding discounting temporal operators. The logic is actually a family of logics, each parameterized by a set $\mathcal{D}$ of discounting functions.

Let $\mathbb{N} = \{0, 1, \ldots\}$. A function $\eta : \mathbb{N} \to [0, 1]$ is a *discounting function* if $\lim_{i\to\infty} \eta(i) = 0$, and η is strictly monotonic-decreasing. Examples for natural discounting functions are $\eta(i) = \lambda^i$ for some $\lambda \in (0, 1)$, and $\eta(i) = \frac{1}{i+1}$. Note that the strict monotonicity implies that $\eta(i) > 0$ for all $i \in \mathbb{N}$.

Given a set of discounting functions $\mathcal{D}$, we define the logic $\text{LTL}^{\text{disc}}[\mathcal{D}]$ as follows. The syntax of $\text{LTL}^{\text{disc}}[\mathcal{D}]$ adds to LTL a *discounting-Until* operator $\varphi \mathsf{U}_\eta \psi$ for every function $\eta \in \mathcal{D}$. Thus, a $\text{LTL}^{\text{disc}}[\mathcal{D}]$ formula is one of the following:

—$\texttt{True}$, or p, for $p \in AP$.
—$\neg\varphi_1$, $\varphi_1 \vee \varphi_2$, $\mathsf{X}\varphi_1$, $\varphi_1 \mathsf{U} \varphi_2$, or $\varphi_1 \mathsf{U}_\eta \varphi_2$, for $\text{LTL}^{\text{disc}}[\mathcal{D}]$ formulas φ_1 and φ_2, and a function $\eta \in D$.

Recall that a logic in the family LTL[$\mathcal{F}$] need not have functions that correspond to the usual Boolean operators; in particular, $\mathcal{F}$ need not contains negation. On the other hand, the logic $\text{LTL}^{\text{disc}}[\mathcal{D}]$ does include the Boolean operators $\neg$ and $\vee$.

The semantics of $\text{LTL}^{\text{disc}}[\mathcal{D}]$ is defined with respect to a *computation* $\pi = \pi_0, \pi_1, \ldots \in (2^{AP})^\omega$. Given a computation π and an $\text{LTL}^{\text{disc}}[\mathcal{D}]$ formula φ, the truth value of φ in π is a value in $[0, 1]$, denoted $[\![\pi, \varphi]\!]$. The value is defined by induction on the structure of φ as described in Table II, where $\pi^i = \pi_i, \pi_{i+1}, \ldots$

The intuition of the discounted-until operator is that events that happen in the future have a lower influence, and the rate by which this influence decreases depends on the function η.[7] For example, the satisfaction value of a formula $\varphi \mathsf{U}_\eta \psi$ in a computation π depends on the best (supremum) value that ψ can get along the entire computation, while considering the discounted satisfaction of ψ at a position i, as a result of multiplying it by $\eta(i)$, and the same for the value of φ in the prefix leading to the ith position.

[7]Observe that, in our semantics, the satisfaction value of future events tends to 0. One may think of scenarios in which future events are discounted towards another value in $[0, 1]$ (e.g., discounting towards $\frac{1}{2}$ as ambivalence regarding the future). We address this in Section 5.3.

Table II. The Semantics of $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$

Formula	Satisfaction value
$[\![\pi, \mathtt{True}]\!]$	1
$[\![\pi, p]\!]$	$1 \quad \text{if } p \in \pi_0$ $0 \quad \text{if } p \notin \pi_0$
$[\![\pi, \neg\varphi_1]\!]$	$1 - [\![\pi, \varphi_1]\!]$
$[\![\pi, \varphi_1 \vee \varphi_2]\!]$	$\max\{[\![\pi, \varphi_1]\!], [\![\pi, \varphi_2]\!]\}$
$[\![\pi, \mathsf{X}\varphi_1]\!]$	$[\![\pi^1, \varphi_1]\!]$
$[\![\pi, \varphi_1 \mathsf{U} \varphi_2]\!]$	$\sup_{i\geq 0}\{\min\{[\![\pi^i, \varphi_2]\!], \min_{0\leq j<i}\{[\![\pi^j, \varphi_1]\!]\}\}\}$
$[\![\pi, \varphi_1 \mathsf{U}_\eta \varphi_2]\!]$	$\sup_{i\geq 0}\{\min\{\eta(i)[\![\pi^i, \varphi_2]\!], \min_{0\leq j<i}\{\eta(j)[\![\pi^j, \varphi_1]\!]\}\}\}$

We add the standard abbreviations $\mathsf{F}\varphi \equiv \mathtt{True}\mathsf{U}\varphi$ and $\mathsf{G}\varphi = \neg\mathsf{F}\neg\varphi$, as well as their quantitative counterparts: $\mathsf{F}_\eta\varphi \equiv \mathtt{True}\mathsf{U}_\eta\varphi$, and $\mathsf{G}_\eta\varphi = \neg\mathsf{F}_\eta\neg\varphi$. Note that $[\![\pi, \mathsf{F}_\eta\varphi]\!] = \sup_{i\geq 0}\{\min\{\eta(i)[\![\pi^i, \varphi]\!], \min_{0\leq j<i}\{\eta(j)\cdot 1\}\}\}$. Since η is decreasing and $i > j$, the latter becomes $\sup_{i\geq 0}\{\eta(i)[\![\pi^i, \varphi]\!]\}$. From this, we also get $[\![\pi, \mathsf{G}_\eta\varphi]\!] = \inf_{i\geq 0}\{1 - \eta(i)(1 - [\![\pi^i, \varphi]\!])\}$.

Remark 4.1. A more restricted way of future-discounting can be captured with a *discounted* X *operator* X_λ where $[\![\pi, \mathsf{X}_\lambda\varphi]\!] = \lambda[\![\pi^1, \varphi]\!]$. In Section 6.2, we show how the X_λ operator can be expressed in $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$, without adding it explicitly.

A computation of the form $\pi = u \cdot v^\omega$, for $u, v \in (2^{AP})^*$, with $v \neq \epsilon$, is called a *lasso computation*. We observe that, since a specific lasso computation has only finitely many distinct suffixes, the sup in the semantics of $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$ can be replaced with max when applied to lasso computations. For the semantics of $\varphi\mathsf{U}\psi$, this is trivial. For the semantics of $\varphi\mathsf{U}_\eta\psi$, recall that $[\![\pi, \varphi_1\mathsf{U}_\eta\varphi_2]\!] = \sup_{i\geq 0} H$, where H is the term $\min\{\eta(i)[\![\pi^i, \varphi_2]\!], \min_{0\leq j<i}\{\eta(j)[\![\pi^j, \varphi_1]\!]\}\}$. Since η is decreasing, it follows that, for a large enough i_0, H is decreasing as a function of $i \geq i_0$. Thus, the sup is attained within a finite prefix, therefore is a max.

The semantics of $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$ is extended to *Kripke structures* by taking the path that admits the lowest satisfaction value. Formally, for a Kripke structure $\mathcal{K}$ and an $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$ formula φ, we have that $[\![\mathcal{K}, \varphi]\!] = \inf\{[\![\pi, \varphi]\!] : \pi \text{ is a computation of } \mathcal{K}\}$.

Example 4.2. Consider a lossy-disk: every moment in time there is a chance that some bit would flip its value. Fixing flips is done by a global error-correcting procedure. This procedure manipulates the entire content of the disk, such that, initially, it causes more errors in the disk; the longer it runs, however, the more bits it fixes.

Let *init* and *terminate* be atomic propositions indicating when the error-correcting procedure is initiated and terminated, respectively. The quality of the disk (i.e., a measure of the amount of correct bits) can be specified by the formula $\varphi = \mathsf{GF}_\eta(\mathit{init} \wedge \neg\mathsf{F}_\mu\mathit{terminate})$ for some appropriate discounting functions η and μ. Intuitively, φ gets a higher satisfaction value the shorter the waiting time is between initiations of the error-correcting procedure, and the longer the procedure runs (i.e., not terminated) in between these initiations. Note that the "worst-case" nature of $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$ fits here. For instance, running the procedure for a very short time, even once, will cause many errors.

4.2. Translating $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$ to Automata

We start by translating a given $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$ formula φ and a threshold v to an alternating weak automaton $\mathcal{A}_{\varphi,v}$ such that $L(\mathcal{A}_{\varphi,v}) \neq \emptyset$ if and only if there exists a computation π such that $[\![\pi, \varphi]\!] > v$. The challenge here is that φ has infinitely many satisfaction

values, naïvely implying an infinite-state automaton. We show that, using the threshold and the discounting behavior of the eventualities, we can restrict attention to a finite resolution of satisfaction values, enabling the construction of a finite automaton. Complexity-wise, the size of $\mathcal{A}_{\varphi,v}$ depends on the functions in $\mathcal{D}$. In Section 4.4, we analyze the complexity for the case of exponential-discounting functions.

The second step is to construct a nondeterministic Büchi automaton $\mathcal{B}$ that is equivalent to $\mathcal{A}_{\varphi,v}$. In general, alternation removal might involve an exponential blowup in the state space [Miyano and Hayashi 1984]. We show, by a careful analysis of $\mathcal{A}_{\varphi,v}$, that we can remove its alternation, ending up with a nondeterministic automaton that is only exponential in φ.

4.2.1. Alternating Weak Automata. For a given set X, let $\mathcal{B}^+(X)$ be the set of positive Boolean formulas over X (i.e., Boolean formulas built from elements in X using $\wedge$ and $\vee$), where we also allow the formulas `True` and `False`. For $Y \subseteq X$, we say that Y *satisfies* a formula $\theta \in \mathcal{B}^+(X)$ if and only if the truth assignment that assigns *true* to the members of Y and assigns *false* to the members of $X \setminus Y$ satisfies θ. An *alternating Büchi automaton on infinite words* is a tuple $\mathcal{A} = \langle \Sigma, Q, q_{in}, \delta, \alpha \rangle$, where Σ is the input alphabet, Q is a finite set of states, $q_{in} \in Q$ is an initial state, $\delta : Q \times \Sigma \to \mathcal{B}^+(Q)$ is a transition function, and $\alpha \subseteq Q$ is a set of accepting states. We define runs of $\mathcal{A}$ by means of (possibly) infinite DAGs (directed acyclic graphs). A run of $\mathcal{A}$ on a word $w = \sigma_0 \cdot \sigma_1 \cdots \in \Sigma^\omega$ is a (possibly) infinite DAG $\mathcal{G} = \langle V, E \rangle$, satisfying the following (note that there may be several runs of $\mathcal{A}$ on w).

—$V \subseteq Q \times \mathbb{N}$ is as follows. Let $Q_l \subseteq Q$ denote all states in level l. Thus, $Q_l = \{q : \langle q, l \rangle \in V\}$. Then, $Q_0 = \{q_{in}\}$, and Q_{l+1} satisfies $\bigwedge_{q \in Q_l} \delta(q, \sigma_l)$.
—For every $l \in \mathbb{N}$, Q_l is minimal with respect to set containment.
—$E \subseteq \bigcup_{l \geq 0} (Q_l \times \{l\}) \times (Q_{l+1} \times \{l+1\})$ is such that, for every state $q \in Q_l$, the set $\{q' \in Q_{l+1} : (\langle q, l \rangle, \langle q', l+1 \rangle) \in E\}$ satisfies $\delta(q, \sigma_l)$.

Thus, the root of the DAG contains the initial state of the automaton, and the states associated with nodes in level $l+1$ satisfy the transitions from states corresponding to nodes in level l. The run $\mathcal{G}$ accepts the word w if all its infinite paths satisfy the acceptance condition α. Thus, in the case of Büchi automata, all the infinite paths have infinitely many nodes $\langle q, l \rangle$ such that $q \in \alpha$ (it is not hard to prove that every infinite path in $\mathcal{G}$ is part of an infinite path starting in level 0). A word w is accepted by $\mathcal{A}$ if there is a run that accepts it. The language of $\mathcal{A}$, denoted $L(\mathcal{A})$, is the set of infinite words that $\mathcal{A}$ accepts.

When the formulas in the transition function of $\mathcal{A}$ contain only disjunctions, then $\mathcal{A}$ is nondeterministic, and its runs are DAGs of width 1, in which, at each level, there is a single node.

The alternating automaton $\mathcal{A}$ is *weak*, denoted AWA, if its state space Q can be partitioned into sets $Q_1, \ldots, Q_k$, such that the following hold. First, for every $1 \leq i \leq k$ either $Q_i \subseteq \alpha$, in which case we say that Q_i is an accepting set, or $Q_i \cap \alpha = \emptyset$, in which case we say that Q_i is rejecting. Second, there is a partial-order $\leq$ over the sets, and for every $1 \leq i, j \leq k$, if $q \in Q_i$, $s \in Q_j$, and $s \in \delta(q, \sigma)$ for some $\sigma \in \Sigma$, then $Q_j \leq Q_i$. Thus, transitions can lead only to states that are smaller in the partial order. Consequently, each run of an AWA eventually gets trapped in a set Q_i and is accepting if and only if this set is accepting.

4.2.2. From $\mathsf{LTL}^{disc}[\mathcal{D}]$ *to* AWA. Intuitively, the states of the AWA that we construct correspond to assertions of the form $\psi > t$ or $\psi < t$ for every subformula ψ of φ, and for certain thresholds $t \in [0, 1]$. A lasso computation π is then accepted from state $\psi > t$

if and only if $[\![\pi, \psi]\!] > t$. We note that the assumption about the computation being a lasso is only needed for the "only if" direction.

The AWA is used in our solution to the model-checking problem by setting the initial state to $\varphi > v$. There, the assumption about the computation being a lasso does not influence the solution's generality since the language of an automaton is nonempty if and only if there is a lasso witness for its nonemptiness.

Defining the appropriate transition function for the AWA follows the semantics of $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$ in the expected manner. A naïve construction, however, yields an infinite-state automaton (even if we only expand the state space on-the-fly, as discounting formulas can take infinitely many satisfaction values). As can be seen in the proof of Theorem 4.6, the "problematic" transitions are those that involve the discounting operators. The key observation is that, given a threshold v and a computation π, when evaluating a discounted operator on π, one can restrict attention to two cases: either the satisfaction value of the formula goes below v, in which case this happens after a bounded prefix, or the satisfaction value always remains above v, in which case we can replace the discounted operator with a Boolean one. This observation allows us to expand only a finite number of states on-the-fly.

Before describing the construction of the AWA, we need the following lemma, which reduces an extreme satisfaction of an $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$ formula, meaning satisfaction with a value of either 0 or 1, to a Boolean satisfaction of an LTL formula.

LEMMA 4.3. *Given an* $\mathrm{LTL}^{disc}[\mathcal{D}]$ *formula* φ*, there exist* LTL *formulas* φ^+ *and* $\varphi^{<1}$ *such that* $|\varphi^+|$ *and* $|\varphi^{<1}|$ *are both in* $O(|\varphi|)$*, and the following hold for every computation* π*.*

(*1*) *If* $[\![\pi, \varphi]\!] > 0$*, then* $\pi \models \varphi^+$*, and if* $[\![\pi, \varphi]\!] < 1$*, then* $\pi \models \varphi^{<1}$*.*
(*2*) *If* π *is a lasso, then if* $\pi \models \varphi^+$*, then* $[\![\pi, \varphi]\!] > 0$*, and if* $\pi \models \varphi^{<1}$*, then* $[\![\pi, \varphi]\!] < 1$*.*

PROOF. We construct φ^+ and $\varphi^{<1}$ by induction on the structure of φ as follows. In all cases but the U case, we do not use the assumption that $\pi = u \cdot v^\omega$ and prove an "if and only if" criterion.

—If φ is of the form `True`, `False`, or p, for an atomic proposition p, then $\varphi^+ = \varphi$ and $\varphi^{<1} = \neg\varphi$. Correctness is trivial.
—If φ is of the form $\psi_1 \vee \psi_2$, then $\varphi^+ = \psi_1^+ \vee \psi_2^+$ and $\varphi^{<1} = \psi_1^{<1} \wedge \psi_2^{<1}$. For every computation π, we have that $[\![\pi, \varphi]\!] > 0$ if and only if either $[\![\pi, \psi_1]\!] > 0$ or $[\![\pi, \psi_2]\!] > 0$, and $[\![\pi, \varphi]\!] < 1$ if and only if both $[\![\pi, \psi_1]\!] < 1$ and $[\![\pi, \psi_2]\!] < 1$.
—If φ is of the form $\mathsf{X}\psi_1$, then $\varphi^+ = \mathsf{X}(\psi_1^+)$ and $\varphi^{<1} = \mathsf{X}(\psi_1^{<1})$. Correctness is trivial.
—If φ is of the form $\psi_1 \mathsf{U} \psi_2$, then $\varphi^+ = \psi_1^+ \mathsf{U} \psi_2^+$ and $\varphi^{<1} = \neg((\neg(\psi_1^{<1}))\mathsf{U}(\neg(\psi_2^{<1})))$.

We start with φ^+. For every computation π, we have that $[\![\pi, \varphi]\!] > 0$ if and only if there exists $i \geq 0$ such that $[\![\pi^i, \psi_2]\!] > 0$ and for every $0 \leq j < i$ it holds that $[\![\pi^j, \psi_1]\!] > 0$. This happens if and only if π satisfies $\psi_1^+ \mathsf{U} \psi_2^+$.

Before we turn to the case of $\varphi^{<1}$, let us note that readers familiar with the release (R) operator of LTL may find it clearer to observe that $\varphi^{<1} = \psi_1^{<1} \mathsf{R} \psi_2^{<1}$, which perhaps gives a clearer intuition for the correctness of the construction.

Now, if $[\![\pi, \varphi]\!] < 1$, then, for every $i \geq 0$, it holds that either $[\![\pi^i, \psi_2]\!] < 1$ or $[\![\pi^j, \psi_1]\!] < 1$ for some $0 \leq j < i$. Thus, for every $i \geq 0$, either $\pi^i \models \psi_2^{<1}$, or $\pi^j \models \psi_1^{<1}$ for some $0 \leq j < i$. Thus, for every $i \geq 0$, either $\pi^i \not\models \neg(\psi_2^{<1})$, or $\pi^j \not\models \neg(\psi_1^{<1})$ for some $0 \leq j < i$. It follows that $\pi \not\models (\neg(\psi_1^{<1}))\mathsf{U}(\neg(\psi_2^{<1}))$. Equivalently, $\pi \models \neg((\neg(\psi_1^{<1}))\mathsf{U}(\neg(\psi_2^{<1})))$.

Conversely, if $\pi = u \cdot v^\omega$ and $\pi \models \neg((\neg(\psi_1^{<1}))\mathsf{U}(\neg(\psi_2^{<1})))$, then $\pi \not\models (\neg(\psi_1^{<1}))\mathsf{U}(\neg(\psi_2^{<1}))$; thus, for every $i \geq 0$, either $\pi^i \models \psi_2^{<1}$ or $\pi^j \models \psi_1^{<1}$ for some $0 \leq j < i$. By the induction hypothesis, for every $i \geq 0$, either $[\![\pi^i, \psi_2]\!] < 1$ or $[\![\pi^j, \psi_1]\!] < 1$ for some $0 \leq j < i$.

We now use the assumption that $\pi = u \cdot v^\omega$ to observe that the sup in the expression for $[\![\pi, \varphi]\!]$ is attained as a max, as there are only finitely many distinct suffixes for π (i.e., $\pi^0, \ldots, \pi^{|u|+|v|-1}$). Thus, since all the elements in the max are strictly smaller than 1, we conclude that $[\![\pi, \varphi]\!] < 1$.

—If $\varphi = \neg\psi$, then $\varphi^+ = \psi^{<1}$ and $\varphi^{<1} = \psi^+$. Again, correctness is trivial.

—If $\varphi = \psi_1 \mathsf{U}_\eta \psi_2$ for $\eta \in \mathcal{D}$, then $\varphi^+ = \psi_1^+ \mathsf{U} \psi_2^+$. Since $\eta(i) > 0$ for all $i \geq 0$, then $\varphi^+ = \psi_1 \mathsf{U} {\psi_2}^+$.

Now, $\varphi^{<1}$ is defined as follows. First, if $\eta(0) < 1$, then $\varphi^{<1} = \texttt{True}$. If $\eta(0) = 1$, then $\varphi^{<1} = {\psi_2}^{<1}$. Since η is strictly decreasing, the only chance of φ to have $[\![\pi, \varphi]\!] = 1$ is when both $\eta(0) = 1$ and $[\![\pi^0, \psi_2]\!] = 1$. Since a satisfaction value cannot exceed 1, the latter happens if and only if $\eta(0) = 1$ and $\pi \not\models \psi_2^{<1}$ (for which the "only if" direction is valid when π is a lasso, as is assumed).

Finally, it is easy to see that $|\varphi^+|$ and $|\varphi^{<1}|$ are both $O(|\varphi|)$. □

Henceforth, given an $\text{LTL}^{\text{disc}}[\mathcal{D}]$ formula φ, we refer to φ^+ as in Lemma 4.3.

Consider an $\text{LTL}^{\text{disc}}[\mathcal{D}]$ formula φ. By Lemma 4.3, if there exists a computation π such that $[\![\pi, \varphi]\!] > 0$, then φ^+ is satisfiable. Conversely, since φ^+ is a Boolean LTL formula, then, by Vardi [1996], we know that φ^+ is satisfiable if and only if there exists a lasso computation π that satisfies it, in which case $[\![\pi, \varphi]\!] > 0$. We thus get the following.

COROLLARY 4.4. *Consider an* $\text{LTL}^{disc}[\mathcal{D}]$ *formula* φ. *There exists a computation* π *such that* $[\![\pi, \varphi]\!] > 0$ *if and only if there exists a lasso computation* π' *such that* $[\![\pi', \varphi]\!] > 0$, *in which case* $\pi' \models \varphi^+$ *as well.*

Remark 4.5. The curious reader may wonder why we do not prove that $[\![\pi, \varphi]\!] > 0$ if and only if $\pi \models \varphi^+$ for every computation π. As it turns out, a translation that is also valid for computations that are not lasso-shaped (i.e., ones with no period) is not always possible. For example, as is the case with the prompt-eventuality operator of Kupferman et al. [2009], the formula $\varphi = \mathsf{G}(\mathsf{F}_\eta p)$ is such that the set of computations π with $[\![\pi, \varphi]\!] > 0$ is not ω-regular; thus, one cannot hope to define an LTL formula φ^+.

Prior to providing the translation, we give some necessary definitions. For a function $f : \mathbb{N} \to [0, 1]$ and for $k \in \mathbb{N}$, we define the function $f^{+k} : \mathbb{N} \to [0, 1]$, for which, for every $i \in \mathbb{N}$, we have that $f^{+k}(i) = f(i + k)$.

Let φ be an $\text{LTL}^{\text{disc}}[\mathcal{D}]$ formula over AP. We define the *extended closure* of φ, denoted $xcl(\varphi)$, to be the set of all the formulas ψ of the following *classes*:

(1) ψ is a subformula of φ.
(2) ψ is a subformula of θ^+ or $\neg\theta^+$, where θ is a subformula of φ.
(3) ψ is of the form $\theta_1 \mathsf{U}_{\eta^{+k}} \theta_2$ for $k \in \mathbb{N}$, where $\theta_1 \mathsf{U}_\eta \theta_2$ is a subformula of φ.

Observe that $xcl(\varphi)$ may be infinite, and that it has both $\text{LTL}^{\text{disc}}[\mathcal{D}]$ formulas (from Classes 1 and 3) and LTL formulas (from Class 2).

THEOREM 4.6. *Given an* $\text{LTL}^{disc}[\mathcal{D}]$ *formula* φ *and a threshold* $v \in [0, 1]$, *there exists an* AWA $\mathcal{A}_{\varphi,v}$ *such that, for every computation* π, *the following hold:*

(*1*) *If* $[\![\pi, \varphi]\!] > v$, *then* $\mathcal{A}_{\varphi,v}$ *accepts* π.
(*2*) *If* $\mathcal{A}_{\varphi,v}$ *accepts* π *and* π *is a lasso computation, then* $[\![\pi, \varphi]\!] > v$.

PROOF. We construct $\mathcal{A}_{\varphi,v} = \langle Q, 2^{AP}, Q_0, \delta, \alpha \rangle$ as follows.

The state space Q consists of two types of states. Type-1 states are assertions of the form $(\psi > t)$ or $(\psi < t)$, where $\psi \in xcl(\varphi)$ is of Class 1 or 3 and $t \in [0, 1]$. Type-2 states correspond to LTL formulas of Class 2. Let S be the set of Type-1 and Type-2 states

for all $\psi \in xcl(\varphi)$ and thresholds $t \in [0, 1]$. Then, Q is the subset of S that is reachable from the initial state via the transition function defined later. We later show that Q is indeed finite.

The initial state of $\mathcal{A}_{\varphi,v}$ is the Type-1 state $(\varphi > v)$. We split the definition of the accepting states and the transition function between Type-2 and Type-1 states. For Type-2 states, we use the standard translation from LTL to AWA [Vardi 1996]. For completeness, we describe the construction here. Readers who are familiar with it can skip to the definitions for Type-1 states.

Type-2 states. Recall that the Type-2 states correspond to subformulas of θ^+ or $\neg\theta^+$, where θ is a subformula of φ. In particular, we get that, for every Type-2 state ψ, the state $\neg\psi$ is also a Type-2 state (we identify $\neg\neg\psi$ with ψ). Let Q_2 be the set of Type-2 states.

Consider a formula $\zeta \in \mathcal{B}^+(Q_2)$. We obtain the *dual formula* $\widetilde{\zeta}$ by switching `True` and `False`, $\wedge$ and $\vee$, and by negating the atoms in Q_2. Formally, we define $\widetilde{\zeta}$ inductively, as follows.

—$\widetilde{q} = \neg q$ for every $q \in Q$.
—$\widetilde{\texttt{True}} = \texttt{False}$ and $\widetilde{\texttt{False}} = \texttt{True}$.
—$\widetilde{\alpha \wedge \beta} = \widetilde{\alpha} \vee \widetilde{\beta}$ and $\widetilde{\alpha \vee \beta} = \widetilde{\alpha} \wedge \widetilde{\beta}$.

The transition function $\delta : Q_2 \times 2^{AP} \to \mathcal{B}^+(Q_2)$ on Type-2 states is defined as follows. Let $\sigma \in 2^{AP}$.

—$\delta(p, \sigma) = \texttt{True}$ if $p \in \sigma$ and $\delta(p, \sigma) = \texttt{False}$ if $p \notin \sigma$.
—$\widetilde{\delta(\psi_1 \vee \psi_2, \sigma)} = \delta(\psi_1, \sigma) \vee \delta(\psi_2, \sigma)$.
—$\delta(\neg\psi, \sigma) = \widetilde{\delta(\psi, \sigma)}$.
—$\delta(\mathsf{X}\psi, \sigma) = \psi$.
—$\delta(\psi_1 \mathsf{U} \psi_2, \sigma) = \delta(\psi_2, \sigma) \vee (\delta(\psi_1, \sigma) \wedge (\psi_1 \mathsf{U} \psi_2))$.

Finally, the accepting Type-2 states are those that correspond to formulas of the form $\neg(\psi_1 \mathsf{U} \psi_2)$. By Vardi [1996], a computation π is accepted from state $q \in Q_2$ if and only if $\pi \models q$.

Type-1 states. The accepting Type-1 states are those of the form $(\psi_1 \mathsf{U} \psi_2 < t)$. The transition function for Type-1 states is defined as follows. Let $\sigma \in 2^{AP}$.

—$\delta((\texttt{True} > t), \sigma) = \begin{cases} \texttt{True} & \text{if } t < 1, \\ \texttt{False} & \text{if } t = 1. \end{cases}$
—$\delta((\texttt{False} > t), \sigma) = \texttt{False}$.
—$\delta((\texttt{True} < t), \sigma) = \texttt{False}$.
—$\delta((\texttt{False} < t), \sigma) = \begin{cases} \texttt{True} & \text{if } t > 0, \\ \texttt{False} & \text{if } t = 0. \end{cases}$
—$\delta((p > t), \sigma) = \begin{cases} \texttt{True} & \text{if } p \in \sigma \text{ and } t < 1, \\ \texttt{False} & \text{otherwise.} \end{cases}$
—$\delta((p < t), \sigma) = \begin{cases} \texttt{False} & \text{if } p \in \sigma \text{ or } t = 0, \\ \texttt{True} & \text{otherwise.} \end{cases}$
—$\delta((\psi_1 \vee \psi_2 > t), \sigma) = \delta((\psi_1 > t), \sigma) \vee \delta((\psi_2 > t), \sigma)$.
—$\delta((\psi_1 \vee \psi_2 < t), \sigma) = \delta((\psi_1 < t), \sigma) \wedge \delta((\psi_2 < t), \sigma)$.
—$\delta((\neg\psi_1 > t), \sigma) = \delta((\psi_1 < 1 - t), \sigma)$.
—$\delta((\neg\psi_1 < t), \sigma) = \delta((\psi_1 > 1 - t), \sigma)$.
—$\delta((\mathsf{X}\psi_1 > t), \sigma) = (\psi_1 > t)$.
—$\delta((\mathsf{X}\psi_1 < t), \sigma) = (\psi_1 < t)$.

$$—\delta((\psi_1 \mathsf{U} \psi_2 > t), \sigma) = \begin{cases} \delta((\psi_2 > t), \sigma) \vee [\delta((\psi_1 > t), \sigma) \wedge (\psi_1 \mathsf{U} \psi_2 > t)] & \text{if } 0 < t < 1, \\ \texttt{False} & \text{if } t = 1, \\ \delta(((\psi_1 \mathsf{U} \psi_2)^+), \sigma) & \text{if } t = 0. \end{cases}$$

$$—\delta((\psi_1 \mathsf{U} \psi_2 < t), \sigma) = \begin{cases} \delta((\psi_2 < t), \sigma) \wedge [\delta((\psi_1 < t), \sigma) \vee (\psi_1 \mathsf{U} \psi_2 < t)] & \text{if } 0 < t \leq 1, \\ \texttt{False} & \text{if } t = 0. \end{cases}$$

$$—\delta((\psi_1 \mathsf{U}_\eta \psi_2 > t), \sigma) = \begin{cases} \delta((\psi_2 > \frac{t}{\eta(0)}), \sigma) \vee [\delta((\psi_1 > \frac{t}{\eta(0)}), \sigma) \wedge (\psi_1 \mathsf{U}_{\eta^{+1}} \psi_2 > t)] & \text{if } 0 < \frac{t}{\eta(0)} < 1, \\ \texttt{False} & \text{if } \frac{t}{\eta(0)} \geq 1, \\ \delta(((\psi_1 \mathsf{U}_\eta \psi_2)^+), \sigma) & \text{if } \frac{t}{\eta(0)} = 0 \text{ (i.e., } t = 0). \end{cases}$$

$$—\delta((\psi_1 \mathsf{U}_\eta \psi_2 < t), \sigma) = \begin{cases} \delta((\psi_2 < \frac{t}{\eta(0)}), \sigma) \wedge [\delta((\psi_1 < \frac{t}{\eta(0)}), \sigma) \vee (\psi_1 \mathsf{U}_{\eta^{+1}} \psi_2 < t)] & \text{if } 0 < \frac{t}{\eta(0)} \leq 1, \\ \texttt{True} & \text{if } \frac{t}{\eta(0)} > 1, \\ \texttt{False} & \text{if } \frac{t}{\eta(0)} = 0 \text{ (i.e., } t = 0). \end{cases}$$

We provide some intuition for the more complex parts of the transition function: consider, for example, the transition $\delta((\psi_1 \mathsf{U}_\eta \psi_2 > t), \sigma)$. Since η is decreasing, the highest possible satisfaction value for $\psi_1 \mathsf{U}_\eta \psi_2$ is $\eta(0)$. Thus, if $\eta(0) \leq t$ (equivalently, $\frac{t}{\eta(0)} \geq 1$), then it cannot hold that $\psi_1 \mathsf{U}_\eta \psi_2 > t$; thus, the transition is to `False`. If $t = 0$, then we only need to ensure that the satisfaction value of $\psi_1 \mathsf{U}_\eta \psi_2$ is not 0. To do so, we require that $(\psi_1 \mathsf{U}_\eta \psi_2)^+$ is satisfied. By Corollary 4.4, this is equivalent to the satisfiability of the former. Thus, the transition is identical to that of the state $(\psi_1 \mathsf{U}_\eta \psi_2)^+$. Finally, if $0 < t < \eta(0)$, then (slightly abusing notation) the assertion $\psi_1 \mathsf{U}_\eta \psi_2 > t$ is true if either $\eta(0)\psi_2 > t$ is true, or both $\eta(0)\psi_1 > t$ and $\psi_1 \mathsf{U}_{\eta+1} \psi_2 > t$ are true.

Note that each path in the run of $\mathcal{A}_{\varphi,v}$ eventually gets trapped in a single state. Thus, $\mathcal{A}_{\varphi,v}$ is indeed an AWA. The intuition behind the acceptance condition is as follows. Getting trapped in state of the form $(\psi_1 \mathsf{U} \psi_2 < t)$ is allowed, as the eventuality is satisfied with value 0. On the other hand, getting stuck in other states (or Type-1) is not allowed, as they involve eventualities that are not satisfied in the threshold promised for them.

This concludes the definition of $\mathcal{A}_{\varphi,v}$.

We now show that $\mathcal{A}_{\varphi,v}$ is indeed finite and correct.

Intuitively, we observe that, while the construction as described earlier is infinite (indeed, uncountable), only finitely many states are reachable from the initial state $(\varphi > v)$, and we can compute these states in advance. This follows from the fact that once the proportion between t and $\eta(i)$ goes above 1, for Type-1 states associated with threshold t and subformulas with a discounting function η, we do not have to generate new states.

We start with some notations. For every state $(\psi > t)$ (resp., $(\psi < t)$) we refer to ψ and t as the state's *formula* and *threshold*, respectively. If the outermost operator in ψ is a discounting operator, then we refer to its discounting function as the state's *discounting function*. For states of Type-2, we refer to their *formula* only (as there is no threshold).

We continue with a couple of observations regarding the structure of $\mathcal{A}_{\varphi,v}$. First, observe that the only cycles in $\mathcal{A}_{\varphi,v}$ are self-loops. Indeed, consider a transition from state q to state $s \neq q$. Let ψ_q, ψ_s be the formulas of q and s, respectively. Going over the different transitions, one may see that ψ_s is a strict subformula of ψ_q, s is a Type-2 state, or both ψ_q and ψ_s have an outermost discounting operator with discounting functions

η and η^{+1}, respectively. By induction over the construction of φ, this observation proves that there are only self-cycles in $\mathcal{A}_{\varphi_v}$.

Second, observe that, in every run of $\mathcal{A}_{\varphi,v}$ on an infinite word w, every infinite branch (i.e., a branch that does not reach `True`) must eventually be in a state of the form $\psi_1 \mathsf{U} \psi_2 > t$, $\psi_1 \mathsf{U} \psi_2 < t$, $\psi_1 \mathsf{U} \psi_2$ or $\neg(\psi_1 \mathsf{U} \psi_2)$ (if it is a Type-2 state). Indeed, these states are the only states that have a self-loop, and the only cycles in the automaton are self-loops.

We now prove that there are finitely many states in the construction. First, there are $O(|\psi|)$ Type-2 states for every Boolean formula in $xcl(\varphi)$; thus, there are only finitely many Type-2 states. This follows immediately from Lemma 4.3 and from the construction of an AWA from an LTL formula.

Next, observe that the number of possible state formulas, up to differences in the discounting function, is $O(|\varphi|)$. This is simply the standard closure of φ. It remains to prove that the number of possible thresholds and discounting functions is finite.

We start by claiming that, for every threshold $t > 0$, there are only finitely many reachable states with threshold t. For every discounting function $\eta \in \mathcal{D}$ (that appears in φ), let $i_{t,\eta} = \max\{i : \frac{t}{\eta(i)} \leq 1\}$. The value of $i_{t,\eta}$ is defined, since the functions tend to 0. Observe that, in every transition from a state with threshold t, if the next state is also with threshold t, then the discounting function (if relevant) is either some $\eta' \in \mathcal{D}$, or η^{+1}. There are only finitely many functions of the former kind. As for the latter kind, after taking η^{+1} $i_{t,\eta}$ times, we have that $t/\eta^{+i_{t,\eta}}(0) > 1$. By the definition of δ, in this case, the transitions are to either `True` or `False`. We conclude that, for every threshold, there are only finitely many reachable states with this threshold.

Next, we claim that there are only finitely many reachable thresholds. This follows immediately from the earlier claim. We start from the state $\varphi > v$. From this state, there are only finitely many reachable discounting functions. The next threshold that can be encountered is either $1 - v$ or $\frac{v}{\eta(0)}$ for η that is either in $\mathcal{D}$ or one of the η^{+i} for $i \leq i_{v,\eta}$. Thus, there are only finitely many such thresholds. Further observe that if a different threshold is encountered, then, by the definition of δ, the state's formula is deeper in the generating tree of φ. Thus, there are only finitely many times that a threshold can change along a single path. Thus, by induction over the depth of the generating tree, we can conclude that there are only finitely many reachable thresholds.

We conclude that the number of states of the automaton is finite.

Next, we prove the correctness of the construction. From Lemma 4.3 and the correctness of the standard translation of LTL to AWA, it remains to prove that for every path π and for every state $(\psi > v)$ (resp., $(\psi < v)$):

(1) If $[\![\pi, \psi]\!] > v$ (resp., $[\![\pi, \psi]\!] < v$), then π is accepted from $(\psi > v)$ (resp., $(\psi < v)$).
(2) If $\pi = u \cdot v^\omega$ and π is accepted from state $(\psi > v)$ (resp., $(\psi < v)$), then $[\![\pi, \psi]\!] > v$ (resp., $[\![\pi, \psi]\!] < v$).

The proof is by induction over the construction of φ, and is fairly trivial given the definition of δ. □

Since $\mathcal{A}_{\varphi,v}$ is a Boolean automaton, then its language is not empty if and only if it accepts a lasso computation. Combining this observation with Theorem 4.6, we conclude with the following.

COROLLARY 4.7. *For an* LTL$^{disc}[\mathcal{D}]$ *formula* φ *and a threshold* $v \in [0, 1]$*, it holds that* $L(\mathcal{A}_{\varphi,v}) \neq \emptyset$ *if and only if there exists a computation* π *such that* $[\![\pi, \varphi]\!] > v$.

4.2.3. From $\mathcal{A}_{\varphi,v}$ to an NBA. Every AWA can be translated to an equivalent NBA, yet the state blowup might be exponential [Miyano and Hayashi 1984; Boker et al. 2010].

By carefully analyzing the AWA $\mathcal{A}_{\varphi,v}$ generated in Theorem 4.6, we show that it can be translated to an NBA ending up with a nondeterministic automaton that is only exponential in φ.

The idea behind our complexity analysis is as follows. Translating an AWA to an NBA involves alternation removal, which proceeds by keeping track of entire levels in a run-DAG. Thus, a run of the NBA corresponds to a sequence of subsets of Q. The key to the reduced state space is that the number of such subsets is only $|Q|^{O(|\varphi|)}$ and not $2^{|Q|}$. To see why, consider a subset S of the states of $\mathcal{A}$. We say that S is *minimal* if it does not include two states of the form $\varphi < t_1$ and $\varphi < t_2$ for $t_1 < t_2$, nor two states of the form $\varphi \mathsf{U}_{\eta^{+i}} \psi < t$ and $\varphi \mathsf{U}_{\eta^{+j}} \psi < t$ for $i < j$, and similarly for ">". Intuitively, sets that are not minimal hold redundant assertions, and can be ignored. Accordingly, we restrict the state space of the NBA to have only minimal sets.

LEMMA 4.8. *For an* LTL$^{disc}[\mathcal{D}]$ *formula* φ *and* $v \in [0,1]$*, the* AWA $\mathcal{A}_{\varphi,v}$ *constructed in Theorem 4.6 with state space* Q *can be translated to an* NBA *with* $|Q|^{O(|\varphi|)}$ *states.*

PROOF. Consider the AWA $\mathcal{A}$ obtained from φ using the construction of Section 4.2.2.

In the translations of AWA to NBA using the method of Gastin and Oddoux [2001], the AWA is translated to an NGBA (see Section 2.4) whose states are the subset-construction of the AWA. This gives an exponential blowup in the size of the automaton. We claim that, in our translation, we can, in a sense, avoid this blowup.

Intuitively, each state in the NGBA corresponds to a conjunction of states of the AWA. Consider such a conjunction of states of $\mathcal{A}$. If the conjunction contains two states $(\psi < t_1)$ and $(\psi < t_2)$, and we have that $t_1 < t_2$, then, by the correctness proof of Theorem 4.6, it holds that a path π is accepted from both states, if and only if π is accepted from $(\psi < t_1)$. Thus, in every conjunction of states from $\mathcal{A}$, there is never a need to consider a formula with two different "<" thresholds. Dually, every formula can appear with at most one ">" threshold.

Next, consider conjunctions that contain states of the form $(\psi_1 \mathsf{U}_\eta \psi_2 < t)$ and $(\psi_1 \mathsf{U}_{\eta^{+k}} \psi_2 < t)$. Again, since the former assertion implies the latter, there is never a need to consider two such formulas. Similar observations hold for the other discounting operators.

Thus, we can restrict the construction of the NGBA to states that are conjunctions of states from the AWA, such that no discounting operator appears with two different "offsets."

Further observe that, by the construction of the AWA, the threshold of a discounting formula does not change with the transition to the same discounting formula; only the offset changes. That is, from the state $(\psi_1 \mathsf{U}_\eta \psi_2 < t)$, every reachable state whose formula is $\psi_1 \mathsf{U}_{\eta^{+k}} \psi_2$ has threshold t as well. Accordingly, the possible number of thresholds that can appear with the formula $\psi_1 \mathsf{U}_\eta \psi_2$ in the subset construction of $\mathcal{A}$ is the number of times that this formula appears as a subformula of φ, which is $O(|\varphi|)$.

We conclude that each state of the obtained NGBA is a function that assigns each subformula[8] of φ two thresholds. The number of possible thresholds and offsets is linear in the number of states of $\mathcal{A}$; thus, the number of states of the NGBA is $|\mathcal{A}|^{O(|\varphi|)}$.

Finally, translating the NGBA to an NBA requires multiplying the size of the state space by $|\mathcal{A}|$; thus, the size of the obtained NBA is also $|\mathcal{A}|^{O(|\varphi|)}$. □

[8]Where a subformula may have several occurrences, for example, in the formula $p \wedge \mathsf{X}p$, we have two occurrences of the subformula p.

4.3. Solving the Questions for LTL$^{\text{disc}}[\mathcal{D}]$

The verification and synthesis questions in the quantitative setting are search problems, asking for the best or worst value (see Section 2.2). When referring to a specific threshold, these questions induce decision problems. For example, the model-checking decision problem is to decide, given a system $\mathcal{K}$, a specification φ, and a threshold v, whether $[\![\mathcal{K}, \varphi]\!] \geq v$.

In the case of LTL$[\mathcal{F}]$, in which every formula only has exponentially many satisfaction values, solutions to the decision problems imply solutions to the search problems. On the other hand, an LTL$^{\text{disc}}[\mathcal{D}]$ formula might have infinitely many satisfaction values. Hence, one cannot reduce an LTL$^{\text{disc}}[\mathcal{D}]$ search problem to a finite set of decision problems with respect to specific thresholds. Later, we solve the decision problems for LTL$^{\text{disc}}[\mathcal{D}]$, leaving the search problems open. Moreover, note that, in our construction in Theorem 4.6, we only consider strict inequalities. Therefore, our solutions to the threshold problems are often only for strict inequalities, or, in dual problems, only for nonstrict inequalities. This is reflected also in the solution to the realizability and synthesis problems, when we require a strict inequality to hold in order to infer nonstrict inequality.

The difficulty in handling nonstrict thresholds stems from the lack of a lasso-witness (see Remark 4.5). We note that similar problems are known to be notoriously difficult, and are encountered in other settings: in multiobjective MDPs [Chatterjee et al. 2013], the need for infinite-memory strategies renders certain problems open, and in Boker et al. [2015], the discounted-sum problem remains open when considering words that are not eventually periodic.

—**Satisfiability and validity.** The NBA obtained in Lemma 4.8 can be directly used for solving the strict threshold-satisfiability and strict threshold-validity problems: given an LTL$^{\text{disc}}[\mathcal{D}]$ formula φ and a threshold $v \in [0, 1]$, we can decide whether there is a computation π such that $[\![\pi, \varphi]\!] \sim v$ for $\sim \in \{<, >\}$, and return such a computation when the answer is positive. This is done by simply deciding whether there exists a word that is accepted by the NBA.

The validity problem can also be viewed dually, asking whether $[\![\pi, \varphi]\!] \geq v$ for every computation $\pi \in (2^{AP})^{\omega}$.

Note that solving the nonstrict version of the problems remains an open problem.

—**Implication and equivalence.** In Section 2.5, we solve the implication problem for LTL$[\mathcal{F}]$ by combining the given formulas using the average operator. In the context of LTL$^{\text{disc}}[\mathcal{D}]$, introducing the average operator may lead to undecidability (as we show in Section 6.1). We thus leave this problem open.

—**Model checking.** We solve the model-checking problem by composing the given Kripke structure with the automaton that is obtained from the negation of specification, as done in the traditional automata-based model-checking procedure [Vardi and Wolper 1994]. Consider a Kripke structure $\mathcal{K}$, an LTL$^{\text{disc}}[\mathcal{D}]$ formula φ, and a threshold v. Let $\mathcal{A}_{\neg\varphi,1-v}$ be the NBA obtained from $\neg\varphi$, as defined in Section 4.2. By checking the emptiness of the intersection of $\mathcal{K}$ with $\mathcal{A}_{\neg\varphi,1-v}$, we can solve the threshold model-checking problem. Indeed, $L(\mathcal{A}_{\neg\varphi,1-v}) \cap L(\mathcal{K}) \neq \emptyset$ if and only if there exists a lasso computation π that is induced by $\mathcal{K}$ such that $[\![\pi, \varphi]\!] < v$, which happens if and only if it is not true that $[\![\mathcal{K}, \varphi]\!] \geq v$.

The complexity of the model-checking procedure depends on the discounting functions in $\mathcal{D}$. Intuitively, the faster the discounting tends to 0, the fewer states $\mathcal{A}_{\varphi,v}$ has. For the set of exponential-discounting functions $\boldsymbol{E}$, we provide concrete complexities, showing that it stays in the same complexity classes of standard LTL model checking (Section 4.4).

Finally, similarly to the case of satisfiability, we leave open the strict model-checking problem.

—**Realizability and synthesis.** We provide here a partial solution to the decision problems induced from the realizability and synthesis questions when referring to a specific threshold. Consider an $\text{LTL}^{\text{disc}}[\mathcal{D}]$ formula φ, and assume a partition of the atomic propositions in φ to input and output signals. Given a threshold $v \geq 0$, we can use the NBA $\mathcal{A}_{\varphi,v}$ in order to address the realizability and synthesis problems, as stated in the following theorem.

THEOREM 4.9. *Consider an* $\text{LTL}^{disc}[\mathcal{D}]$ *formula* φ *over* $I \cup O$. *If there exists an* I/O*-transducer, all of whose computations* π *satisfy* $[\![\pi, \varphi]\!] > v$, *then we can generate a finite-state* I/O*-transducer, all of whose computations* τ *satisfy* $[\![\tau, \varphi]\!] \geq v$.

PROOF. Recall that if π is a computation such that $[\![\pi, \varphi]\!] > v$, then $\mathcal{A}_{\varphi,v}$ accepts π. The converse, however, is not true. Still, by carefully examining the construction in Theorem 4.2.2, we observe that, if $\mathcal{A}_{\varphi,v}$ accepts a computation π, then $[\![\pi, \varphi]\!] \geq v$ (note the nonstrict inequality).

Assume a partition of the letters in AP to input and output signals, denoted I and O, respectively. By following standard (Boolean) procedures for synthesis (see Pnueli and Rosner [1989]), we can generate from $\mathcal{A}_{\varphi,v}$ a deterministic tree automaton $\mathcal{D}$ that accepts a 2^O-labeled 2^I-tree if and only if all the paths along the tree are accepted in $\mathcal{A}_{\varphi,v}$. Moreover, it is shown in Gurevich and Harrington [1982] that, if $L(\mathcal{D}) \neq \emptyset$, then $\mathcal{D}$ accepts a regular tree, which is induced by a finite-state transducer. A transducer that induces an accepted tree realizes φ with value at least v. Accordingly, if there exists a transducer $\mathcal{T}$, all of whose computations satisfy $[\![\pi, \varphi]\!] > v$, then the regular tree induced by it is accepted in $\mathcal{D}$. Thus, the language of $\mathcal{D}$ is nonempty, and a witness to its nonemptiness is a regular tree that induced by a transducer all whose computations satisfy $[\![\pi, \varphi]\!] \geq v$. □

We note that this solution is only partial, as there might be an I/O-transducer, all of whose computations π satisfy $[\![\pi, \varphi]\!] \geq v$, but we would not find it. For example, consider the formula $\varphi = p \wedge \neg p$, where p is an input signal. Since every formula is realizable with threshold 0, so is φ. If, however, we consider the automaton $\mathcal{A}_{\varphi,0}$, we get that $L(\mathcal{A}_{\varphi,0}) = \emptyset$. Thus, proceeding to find a transducer from $\mathcal{A}_{\varphi,0}$ results in an answer that no such transducer exists.

The difficulty in solving the issue is similar to the difficulty in solving the satisfiability threshold problem for the case of nonstrict inequality.

Remark 4.10. While we do not solve the search problems with an exact value, our solutions do provide an approximation scheme for the search problems, up to any desired constant: Let $\epsilon > 0$, and consider, for example, the problem of finding the maximal satisfaction value of a formula φ. We solve the problem by doing a binary search on $[0, 1]$; for every threshold v, we check whether $L(\mathcal{A}_{\varphi,v}) \neq \emptyset$. After $\log(\frac{1}{\epsilon})$ iterations, we would have two thresholds $v_1 < v_2$ such that $v_2 - v_1 \leq \epsilon$, and $L(\mathcal{A}_{\varphi,v_1}) \neq \emptyset$ while $L(\mathcal{A}_{\varphi,v_2}) = \emptyset$. This implies that the maximal satisfaction value of φ lies in $[v_1 - \epsilon, v_1 + \epsilon]$. Nakagawa and Hasuo [2015] use a similar scheme in order to synthesize $\text{LTL}^{\text{disc}}[\mathcal{D}]$ specifications with an approximative maximal satisfaction value.

4.4. $\text{LTL}^{\text{disc}}[E]$: The Instantiation of $\text{LTL}^{\text{disc}}[\mathcal{D}]$ with Exponential Discounting

The logic $\text{LTL}^{\text{disc}}[\mathcal{D}]$ provides a general framework for specifying temporal quality and allows for arbitrary discounting functions within the specification formulas. Thus, the

complexities of solving the decision problems for $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$ depend on the choice of discounting functions.

The class of *exponential-discounting* functions is perhaps the most common class of discounting functions, as it describes what happens in many natural processes (e.g., temperature change, capacitor charge, and effective interest rate) [Shapley 1953; De Alfaro et al. 2003]. Formally, for a parameter $\lambda \in (0, 1)$, we define the *exponential-discounting* function $\exp_\lambda : \mathbb{N} \to [0, 1]$ by $\exp_\lambda(i) = \lambda^i$. Let $E = \{\exp_\lambda : \lambda \in (0, 1) \cap \mathbb{Q}\}$, and consider the logic $\mathrm{LTL}^{\mathrm{disc}}[E]$.

In this section, we analyze the complexities of solving the decision problems for $\mathrm{LTL}^{\mathrm{disc}}[E]$, and show that they stay in the same complexity classes of standard LTL.

For an $\mathrm{LTL}^{\mathrm{disc}}[E]$ formula φ, let $F(\varphi)$ be the set $\{\lambda :$ the operator $\mathsf{U}_{\exp_\lambda}$ appears in $\varphi\}$. That is, $F(\varphi)$ is the set of discounting factors that appear in φ. Let $|\langle\varphi\rangle|$ be the length of the description of φ. That is, in addition to $|\varphi|$, we include in $|\langle\varphi\rangle|$ the length, in bits, of describing $F(\varphi)$. Let $|\langle v\rangle|$ be the length of the description of a threshold v.

The core step in our solution to the decision problems with respect to an $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$ formula φ and a threshold v is the generation of an AWA $\mathcal{A}_{\varphi,v}$ (Section 4.2). We will show later that, for $\mathrm{LTL}^{\mathrm{disc}}[E]$ formulas, we can generate $\mathcal{A}_{\varphi,v}$ and reason about it in PSPACE with respect to $|\langle\varphi\rangle|$ and $|\langle v\rangle|$.

We start by showing that the number of states in $\mathcal{A}_{\varphi,v}$ is singly exponential in the descriptions of φ and v (Section 4.4.1). One could have hoped that this will allow for a polynomial description of the states, using a binary representation. It turns out, however, that this is not enough, as the values of the thresholds that appear in the states may be doubly exponential. Nevertheless, we show how to represent and manipulate the thresholds succinctly, using arithmetic circuits rather than binary representations (Section 4.4.2). We conclude with the resulting complexities of the decision problems, among which model checking is shown to be in PSPACE (Section 4.4.3).

4.4.1. The Number of States in $\mathcal{A}_{\varphi,v}$. The number of states in the AWA generated as per Theorem 4.6 depends on the discounting functions. In the following lemma, we show that, for $\mathrm{LTL}^{\mathrm{disc}}[E]$ formulas, it is singly exponential in $|\langle\varphi\rangle|$ and $|\langle v\rangle|$.

LEMMA 4.11. *Given an* $\mathrm{LTL}^{disc}[E]$ *formula* φ *and a threshold* $v \in [0, 1] \cap \mathbb{Q}$*, there exists an* AWA $\mathcal{A}_{\varphi,v}$ *such that, for every computation* π*, the following hold:*

(*1*) *If* $[\![\pi, \varphi]\!] > v$*, then* $\mathcal{A}_{\varphi,v}$ *accepts* π.
(*2*) *If* $\mathcal{A}_{\varphi,v}$ *accepts* π *and* π *is a lasso computation, then* $[\![\pi, \varphi]\!] > v$.

Furthermore, the number of states of $\mathcal{A}_{\varphi,v}$ *is singly exponential in* $|\langle\varphi\rangle|$ *and* $|\langle v\rangle|$.

PROOF. We construct an AWA $\mathcal{A}_{\varphi,v}$ as per Section 4.2.2, with some changes. Recall that the "interesting" states in $\mathcal{A}$ are those of the form $\psi_1 \mathsf{U}_{\eta^{+i}} \psi_2$. Observe that, for the function $\exp_\lambda$, it holds that $\exp_\lambda^{+i} = \lambda^i \cdot \exp_\lambda$. Accordingly, we can replace a state of the form $\psi_1 \mathsf{U}_{\exp_\lambda^{+i}} \psi_2 < t$ with the state $\psi_1 \mathsf{U}_{\exp_\lambda} \psi_2 < \frac{t}{\lambda^i}$, as they express the same assertion. Note that $\frac{t}{\lambda^i}$ may be strictly greater than 1. In order to simplify the transitions, we identify, for $t > 1$, every state $\varphi > t$ with `False` and state $\varphi < t$ with `True`. Finally, notice that $\exp_\lambda(0) = 1$. Thus, we can simplify the construction of $\mathcal{A}_{\varphi,v}$ with the following transitions:

Let $\sigma \in 2^{AP}$, then we have that

$$-\!\!-\delta((\psi_1 \mathsf{U}_{\exp_\lambda} \psi_2 > t), \sigma) = \begin{cases} \delta((\psi_2 > t), \sigma) \vee & \\ \quad [\delta((\psi_1 > t), \sigma) \wedge (\psi_1 \mathsf{U}_{\exp_\lambda} \psi_2 > \frac{t}{\lambda})] & \text{if } 0 < t < 1, \\ \texttt{False} & \text{if } t = 1, \\ \delta(((\psi_1 \mathsf{U}_{\exp_\lambda} \psi_2)^+), \sigma) & \text{if } t = 0. \end{cases}$$

$$—\delta((\psi_1 \mathsf{U}_{\exp_\lambda} \psi_2 < t), \sigma) = \begin{cases} \delta((\psi_2 < t), \sigma) \wedge \\ \quad [\delta((\psi_1 < t), \sigma) \vee (\psi_1 \mathsf{U}_{\exp_\lambda} \psi_2 < \frac{t}{\lambda})] & \text{if } 0 < t \leq 1, \\ \texttt{False} & \text{if } t = 0. \end{cases}$$

The correctness and finiteness of the construction follows from Theorem 4.6, with the earlier observation. We now turn to analyze the number of states in $\mathcal{A}_{\varphi,v}$.

For every state $(\psi > t)$ (resp. $(\psi < t)$) we refer to ψ and t as the state's *formula* and *threshold*, respectively. Observe that the number of possible state formulas is $O(|\varphi|)$. The formulas in the states are either in the closure of φ or are of the form ψ^+, where ψ is in the closure of φ. This is because, in the new transitions, we do not carry the offset, but rather change the threshold; thus, the state formula does not change. Note that this includes all Type-2 states as well.

It remains to bound the number of possible thresholds. Consider a state with threshold t and formula ψ. In every succeeding state, the formula can either stay ψ or proceed to a subformula of ψ. When the formula stays ψ, the threshold can either stay t or change to t/λ where $\lambda \in F(\varphi)$ (in the case that $\psi = \psi_1 \mathsf{U}_{\exp_\lambda} \psi_2$), provided that $t/\lambda < 1$. When the formula is replaced by a subformula ψ, the threshold can stay t or may change to $1-t$, in the case that $\psi = \neg\psi_1$, or to a "fresh" threshold, in the case that ψ is a discounted-until formula with a different discount factor. We do not distinguish between the two cases, and simply state that the state formula changes to a *deeper* subformula. Observe that changing to a deeper subformula can occur at most $O(|\varphi|)$ times along a run, since this is the maximal nesting depth of formulas in φ.

Before going into the technical details, we explain our approach. Recall that the discounting factors in $F\varphi$ and the threshold t are all rational. We start by bounding the number of states that are reachable from a state with threshold t_0 without changing to a deeper subformula. We show that this bound depends on the denominator of t_0. We then consider the thresholds of the states that are reached in this manner, and give a bound on their denominators. We then show that repeating this analysis $|\varphi|$ times, which is the maximal number of changes to deeper subformulas, results in a singly exponential number of states.

Without loss of generality, assume that all the discounting factors in $F\varphi$ have the same denominator $\widehat{q}$. By taking the product of the denominators, we can see that the description size of $\widehat{q}$ is linear in that of the original denominators. Thus, this assumption does not change the representation size of φ.

Consider a state with threshold $t_0 = \frac{p_0}{q_0}$. As we mentioned earlier, as long as the state does not change to a deeper subformula, the threshold can change only to $\frac{t_0}{\lambda}$ for $\lambda \in F\varphi$, provided that $\frac{t_0}{\lambda} < 1$. Thus, the number of thresholds that are reachable in this manner is the maximal $i \in \mathbb{N}$ such that $\frac{t_0}{\lambda^i} < 1$. By rearranging, we get $i \leq \log_\lambda(t_0)$. Since we want a bound from above on this value, we note that this expression is maximized when λ is maximal and t_0 is minimal, which happens when $t_0 = \frac{1}{t_0}$ (since $p_0 \geq 1$) and $\lambda = \frac{\widehat{q}-1}{\widehat{q}}$ (recall that all factors in $F\varphi$ have denominator $\widehat{q}$; thus, the largest possible factor is $\frac{\widehat{q}-1}{\widehat{q}}$). From this, we get that the maximal number of states that are reachable from threshold t_0 without changing to a deeper subformula is at most

$$\log_{\frac{\widehat{q}-1}{\widehat{q}}} \left(\frac{1}{q_0}\right) = \frac{\log(1) - \log(q_0)}{\log(\widehat{q}-1) - \log(\widehat{q})} = \frac{\log(q_0)}{\log(\widehat{q}) - \log(\widehat{q}-1)} < \widehat{q} \log q_0.$$

The last equality follows from the fact $\log(x) - \log(x-1) > \frac{1}{x}$ for all $x \geq 0$, which can be proven by simple analysis.

Next, we consider the threshold of the states that can be reached as described earlier. These thresholds are of the form $\frac{t_0}{\lambda^j} = \frac{p_0 \widehat{q}^j}{q_0 p^j}$ for $j \leq \widehat{q} \log q_0$, where $\lambda = \frac{p}{\widehat{q}}$. Since $p < \widehat{q}$, we conclude that the denominator of these thresholds is at most

$$q_0 \cdot \widehat{q}^{\widehat{q} \log q_0} = 2^{\log q_0} \cdot 2^{\widehat{q} \log \widehat{q} \cdot \log q_0} = 2^{\log q_0 \cdot (\widehat{q} \cdot \log \widehat{q} + 1)} = q_0^{(\widehat{q} \cdot \log \widehat{q} + 1)}.$$

To recap, we saw that starting from threshold t_0 with denominator q_0, without changing to a deeper subformula, we can reach at most $\widehat{q} \log q_0$ states, and their denominators are at most $q_0^{(\widehat{q} \cdot \log \widehat{q} + 1)}$.

Now, consider the case in which the path changes to a deeper subformula. Then, either the threshold does not change or it changes from t to $1-t$, or we go to a new discounted-until formula. In all cases, the denominator of the threshold does not change; thus, this analysis can be repeated. Recall that the number of changes to a deeper formula is $n = O(|\varphi|)$. Accordingly, the sequence $a_1, \ldots, a_n$ of maximal denominators of thresholds after changing to a deeper subformula is given by $a_1 = q_0$ and $a_{i+1} = a_i^{(\widehat{q} \cdot \log \widehat{q} + 1)}$ for all $1 \leq i < n$. It is easy to see that this sequence is increasing, and that $a_i = q_0^{((\widehat{q} \cdot \log \widehat{q} + 1)^i)}$.

Hence, the number of states that are reachable from a threshold t_0 with denominator q_0 can be bound by

$$\prod_{i=1}^{n} \widehat{q} \log(a_i) \leq \widehat{q}^n \log^n(a_n) = \widehat{q}^n \log^n \left(q_0^{((\widehat{q} \cdot \log \widehat{q} + 1)^n)} \right) = \widehat{q}^n \, (\widehat{q} \cdot \log \widehat{q} + 1)^{n^2} \log^n q_0.$$

Finally, assuming a binary encoding of the factors, observe that $\widehat{q} = 2^{\log \widehat{q}}$ is singly exponential in the description of φ. Hence, $\widehat{q}^n = 2^{n \log \widehat{q}}$ is also singly exponential. Similar considerations on the other multiplicands in the product imply that the number of states of the automaton is singly exponential. □

4.4.2. Reasoning on $\mathcal{A}_{\varphi,v}$ in PSPACE. The careful reader notices that, while the number of states in $\mathcal{A}_{\varphi,v}$ constructed in Lemma 4.11 is singly exponential in the description of φ and v, the values of the thresholds that appear in the states may be doubly exponential. Our bound on the maximal denominator of the threshold values is $q_0^{(\widehat{q} \log \widehat{q} + 1)^n}$. This suggests that describing the thresholds in $\mathcal{A}_{\varphi,v}$ in binary results in an exponential description for each state, exceeding our targeted PSPACE algorithm.

We will show how to represent the thresholds in $\mathcal{A}_{\varphi,v}$ succinctly using arithmetic circuits. Then, using results from Allender et al. [2009], we are able to compare thresholds to 1 in PSPACE, thus allow on-the-fly construction of $\mathcal{A}_{\varphi,v}$ in PSPACE.

An *arithmetic circuit* is a rooted-DAG whose nodes are labeled by elements in $\{-, +, *, 1\}$, such that the leaves are all labeled by 1, and the internal nodes have fan-in 2 and are labeled by $-$, $+$, and $*$. Given an arithmetic circuit, its value is the value of the root, computed by applying the operation of each internal node on its inputs.

We start with two simple lemmas on arithmetic circuits that will serve us in the threshold representation.

LEMMA 4.12. *Let $k, l \in \mathbb{N}$, and let C_l be an arithmetic circuit of size $|C_l|$ whose value is l. Then, there exists a circuit of size $|C_l| + k$ that computes $l^{(2^k)}$.*

PROOF. By multiplying the output of C_l with itself using a chain of k multiplication gates, we get the result. □

LEMMA 4.13. *For every $i \in \mathbb{N}$, there exists a circuit C_i of size poly$(\log(i))$ that computes i.*

PROOF. Let the binary expansion of i be $b_r b_{r-1} \dots b_0$, where $r = \lceil \log i \rceil$. Then, $i = \sum_{m=0}^{r} b_m 2^m$. Using at most $r-1$ addition gates, we see that it is enough to construct circuits of size at most $poly(m)$ for 2^m for every $0 \leq m \leq r$ (since the b_m are either 0, in which case we do not need a circuit, or 1, in which case we ignore the b_m and just have 2^m).

Let $0 \leq m \leq r$, and consider the binary expansion $d_t d_{t-1} \dots d_0$ of m (where $t = \lceil \log m \rceil$). We have that $2^m = \prod_{s=0}^{t} 2^{d_t \cdot 2^s}$. We ignore elements in the product where $d_s = 0$, since their value is 1. For elements where $d_s = 1$, we can construct by Lemma 4.12 a circuit for $2^{(2^s)}$ of size $s+1$, by first constructing the constant $l = 2$ (in the notations of Lemma 4.12) using a single addition gate. By using at most t multiplication gates, we construct a circuit that computes 2^m of size $poly(\log m)$, which is less than $poly(m)$, and we are done. □

We are now ready to provide the succinct representation of $\mathcal{A}_{\varphi,v}$.

LEMMA 4.14. *There exists a representation of $\mathcal{A}_{\varphi,v}$, the* AWA *constructed as per the proof of Lemma 4.11, such that given a state s and a letter σ, we can compute $\delta(s,\sigma)$ in PSPACE.*

PROOF. We start by showing how to succinctly represent the thresholds in the states of $\mathcal{A}_{\varphi,v}$. Let $v = \frac{p_0}{q_0}$ and $n = |\varphi|$. Recall that a state of $\mathcal{A}_{\varphi,v}$ consists of a formula ψ and a threshold t, and that all the discounting factors in $F\varphi$ have the same denominator $\widehat{q}$. In a successor state, the threshold can change to either $1-t$ or to t/λ, where $\lambda \in F(\varphi)$.

More precisely, by the proof of Lemma 4.11, we can see that every threshold t is obtained from v by applying at most n compositions of the functions $f(t) = 1-t$ and $g_{\lambda^i}(t) = \frac{t}{\lambda^i}$ for $\lambda \in F(\varphi)$ and $i \leq \widehat{q}(\widehat{q}\log\widehat{q}+1)^n \log q_0$. The functions correspond to negation and discounting until, respectively. Then, nesting corresponds to a change to a deeper subformula, which can happen at most n times along a path. Finally, the maximal number of consecutive applications of discounting without changing to a deeper formula is bounded by $\widehat{q}(\widehat{q}\log\widehat{q}+1)^n \log q_0$.

Observe that i here is singly exponential in $|\langle\varphi\rangle|$ and $|\langle v\rangle|$; thus, the binary encoding of i is polynomial. Thus, we can represent every threshold in $\mathcal{A}_{\varphi,v}$ in polynomial size by encoding the composition of functions presented earlier, where encoding f requires a constant number of bits, and encoding g_{λ^i} requires encoding λ (which is part of the encoding of φ) and i, which is polynomial.

For example[9], suppose that we start with the threshold $\frac{1}{1000}$, then apply negation, discount with $\frac{2}{3}$ for 17 steps, and negate again. This is encoded using the sequence "$f(g_{\frac{2}{3}^{17}}(f(\frac{1}{1000}))$."

It remains to show that, using this encoding, we can compute a successor state in PSPACE. Clearly, describing a successor threshold can be done in polynomial time: either by composing another function (i.e., f or g_λ) or incrementing the exponent i in the outermost g_{λ^i}. The difficult case is when the threshold becomes 1 or greater than 1. Then, we must change the state to a Type-2 formula. However, it is not immediate that comparison to 1 can be efficiently computed using our representation.

Thus, we now restrict to solving the following problem: given a function h, which consists of n compositions of the functions f and g_{λ^i} for $\lambda \in F(\varphi)$ and polynomial i, decide whether $h(v) \geq 1$. In order to solve the problem, we translate $h(v)$ to a polynomial-size *arithmetic circuit*, and check in PSPACE whether the circuit represents a number greater than 0, using the results of Allender et al. [2009].

[9]Our example uses a decimal rather than binary encoding. Clearly, one could also use a binary encoding, which incurs only a polynomial overhead.

Note that the value of an arithmetic circuit is an integer, while our setting uses rational numbers. Thus, we must first show how to handle rational numbers. We do so by representing $h(v)$ using two circuits $h_N(v)$ and $h_D(v)$ for the numerator and denominator, respectively. Then, we have that $h(v) \geq 1$ if and only if $h_N(v) \geq h_D(v) > 0$, or $h_N(v) \leq h_D(v) < 0$. Thus, checking whether $h(v) \geq 1$ can be reduced to two tests: first, whether $h_D(v) > 0$, and second, whether $h_N(v) - h_D(v) \geq 0$ (if $h_D(v) > 0$) or whether $h_N(v) - h_D(v) \leq 0$ (if $h_D(v) < 0$).

We construct h_N and h_D inductively, according to the construction of h. We start with the denominator h_D.

—If $h(v) = v$, we need to construct $h_D(v) = q_0$. This can be done using Lemma 4.13.
—If $h(v) = f(h'(v))$, then $h_D(v) = h'_D(v)$; thus, there is nothing to construct.
—If $h(v) = g_{\lambda^i}(h'(v))$, let $\lambda = \frac{p}{q}$. Then, $h_D(v) = q^i \cdot h'_D(v)$. To construct it, we need only to construct a circuit of polynomial size for q^i, then use a multiplication gate between it and the circuit $C_{h'_D}$ for $h'_D(v)$, which has already been constructed inductively. We start by constructing a circuit C_q that computes q, as per Lemma 4.13. Next, let $b_r b_{r-1} \dots b_0$ be the binary expansion of i, then $q^i = \prod_{j=0}^{r} q^{b_j 2^j}$. From Lemma 4.12, each term $q^{b_j 2^j}$ has a circuit of size at most $r + |C_q|$ that computes it (since $b_j \in \{0, 1\}$). In fact, by reusing the same circuit C_q, the total size of all the circuits that compute $q^{b_j 2^j}$ for all $0 \leq j \leq r$ is at most $|C_q| + r^2$. By connecting these circuits using r multiplication gates, we get a circuit of size $|C_q| + r^2 + r$ for q^i. Thus, a circuit for $h_D(v)$ is of size $|C_q| + r^2 + r + |C_{h'_D}|$.

Note that, in each step of the construction, the size of the circuit for h_D increases additively by a factor polynomial in $|\langle\varphi\rangle|$. Thus, the total size of h_D is polynomial in $|\langle\varphi\rangle|$ and $|\langle v\rangle|$.

Next, we turn to construct $h_N(v)$.

—If $h(v) = v$, we have that $h_D(v) = p_0$, and we construct a circuit similarly to $h_D(v)$, using Lemma 4.13.
—If $h(v) = f(h'(v))$, then we have that $h_N(v) = h'_D(v) - h'_N(v)$. Thus, we simply connect circuits for $h'_D(v)$ and $h'_N(v)$ using a subtraction gate. Note that, since $h'_D(v)$ is polynomial in $|\langle\varphi\rangle|$ and $|\langle v\rangle|$, the size of the circuit for h_N remains polynomial in $|\langle\varphi\rangle|$ and $|\langle v\rangle|$.
—If $h(v) = g_{\lambda^i}(h'(v))$, the construction of a circuit for $h_N(v)$ is similar to that of $h_D(v)$.

This concludes the construction of $h_D(v)$ and $h_N(v)$.

In Allender et al. [2009], it is shown that deciding whether the value of a circuit is nonnegative is in PSPACE. As we showed earier, $h(v) \geq 1$ if and only if either $h_D(v) > 0$, and $h_N(v) - h_D(v) \geq 0$, or $h_D(v) < 0$ and $h_N(v) - h_D(v) \leq 0$. Note that all these tests can be easily reduced to deciding nonnegativity of a circuit. For example, checking whether $h_N(v) - h_D(v) \leq 0$ amounts to constructing a circuit for $h_D(v) - h_N(v)$ using a subtraction gate and checking nonnegativity. This concludes the proof. □

4.4.3. Solving the Decision Problems for $\mathsf{LTL}^{disc}[E]$. Using the succinct $\mathcal{A}_{\varphi,v}$, as per Lemmas 4.11 and 4.14, we can construct an equivalent NBA, as per Lemma 4.8, and solve the satisfiability and model-checking problems in PSPACE.

LEMMA 4.15. *For an* $\mathrm{LTL}^{disc}[E]$ *formula* φ *and* $v \in [0, 1]$, *the* AWA $\mathcal{A}_{\varphi,v}$ *constructed in Lemma 4.11 with state space* Q *can be translated to an* NBA $\mathcal{B}_{\varphi,v}$ *with* $|Q|^{O(|\varphi|)}$ *states. Moreover, given a state* s *of* $\mathcal{B}_{\varphi,v}$, *and a letter* σ, *we can compute* $\delta(s, \sigma)$ *in PSPACE.*

PROOF. The construction and size-analysis of $\mathcal{B}_{\varphi,v}$ is identical to that described in Lemma 4.8.

To ensure that transitions are computable in PSPACE, recall that every state s of $\mathcal{B}_{\varphi,v}$ is a subset of the states of $\mathcal{A}_{\varphi,v}$, of size polynomial in $|\varphi|$. Using the succinct representation of $\mathcal{A}_{\varphi,v}$ described in Lemma 4.14, we can succinctly represent such a state s, while preserving the ability to compute successors in PSPACE, by computing the successors of each state $q \in s$. Since s is of polynomial size and the successors of each state are computable in PSPACE, the total complexity of computing the successors remains in PSPACE. □

THEOREM 4.16. *For an* $\text{LTL}^{disc}[E]$ *formula* φ *and a threshold* $v \in [0,1] \cap \mathbb{Q}$*, the problem of deciding whether there exists a computation* π *such that* $[\![\pi, \varphi]\!] > v$ *is in PSPACE in* $|\langle\varphi\rangle|$ *and* $|\langle v\rangle|$.

PROOF. Analogous to the proof of Theorem 4.17. □

THEOREM 4.17. *For a Kripke structure* $\mathcal{K}$*, an* $\text{LTL}^{disc}[E]$ *formula* φ*, and a threshold* $v \in [0,1] \cap \mathbb{Q}$*, the problem of deciding whether* $[\![\mathcal{K}, \varphi]\!] > v$ *is in NLOGSPACE in the number of states of* $\mathcal{K}$*, and in PSPACE in* $|\langle\varphi\rangle|$ *and* $|\langle v\rangle|$.

PROOF. By Lemma 4.15, we can construct an NBA $\mathcal{B}$ corresponding to φ in PSPACE in $|\langle\varphi\rangle|$ and $|\langle v\rangle|$. Hence, we can check the emptiness of the intersection of $\mathcal{K}$ and $\mathcal{B}$ via standard "on-the-fly" procedures, getting the stated complexities. □

Note that the complexity in Theorem 4.17 is only NLOGSPACE in the system, since our solution does not analyze the Kripke structure, but only takes its product with the specification's automaton. This is in contrast to the approach of model-checking temporal logic with (nondiscounting) accumulative values, where, when decidable, involves a doubly exponential dependency on the size of the system [Boker et al. 2014].

5. EXTENDING TEMPORAL QUALITY

In this section, we consider extensions of the temporal-quality setting. Similar to Section 3.1, we start by extending the framework of $\text{LTL}^{\text{disc}}[\mathcal{D}]$ to handle weighted systems (Section 5.1). We continue with extensions of particular interest in the case of temporal quality, namely, adding past operators (Section 5.2), and changing the tendency of discounting (Section 5.3).

5.1. Weighted Systems

A central property of the logic $\text{LTL}^{\text{disc}}[\mathcal{D}]$ is that the verified system need not be weighted in order to get a quantitative satisfaction—the quantitative aspect stems from taking into account the delays in satisfying the requirements. Nevertheless, $\text{LTL}^{\text{disc}}[\mathcal{D}]$ also naturally fits weighted systems, for which the atomic propositions have a value between 0 and 1. (For the definition of weighted systems, see Section 3.1.)

Consider a weighted Kripke structure $\mathcal{K} = \langle AP, S, I, \rho, L\rangle$, an $\text{LTL}^{\text{disc}}[\mathcal{D}]$ formula φ, and a threshold v. It is possible to extend the construction of $\mathcal{A}_{\varphi,v}$, as described in Section 4.2.2, to an alphabet W^{AP}, where W is a set of possible values for the atomic propositions. We only have to adjust the transition for states that correspond to atomic propositions, as follows: for $p \in AP$, $v \in [0,1]$, and $\sigma \in W^{AP}$, we have that

$$—\delta(p > v, \sigma) = \begin{cases} \texttt{True} & \text{if } \sigma(p) > v, \\ \texttt{False} & \text{otherwise.} \end{cases} \qquad —\delta(p < v, \sigma) = \begin{cases} \texttt{True} & \text{if } \sigma(p) < v, \\ \texttt{False} & \text{otherwise.} \end{cases}$$

5.2. $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$ with Past Operators

One of the well-known augmentations of LTL is the addition of the *past operators* $\mathsf{Y}\varphi$ (*Yesterday*) and $\varphi\mathsf{S}\psi$ (*Since*) [Lichtenstein et al. 1985]. These operators enable the specification of exponentially more succinct formulas, while preserving the PSPACE complexity of model checking.

The semantics in the Boolean setting are as follows. For formulas φ, ψ, a computation π, and an index $i \in \mathbb{N}$, we have that $[\![\pi^i, \mathsf{Y}\varphi]\!] = [\![\pi^{i-1}, \varphi]\!]$ if $i > 0$, and `False` otherwise, and $[\![\pi^i, \varphi\mathsf{S}\psi]\!] = \mathtt{True}$ if there exists $0 \leq j \leq i$ such that $\pi^j \models \psi$ and $\pi^k \models \varphi$ for every $j < k \leq i$.

In this section, we add *discounting-past* operators to $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$, and show how to perform model-checking on the obtained logic.

We add the operators $\mathsf{Y}\varphi$, $\varphi\mathsf{S}\psi$, and $\varphi\mathsf{S}_\eta\psi$ (for $\eta \in \mathcal{D}$) to $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$, and denote the extended logic $\mathrm{PLTL}^{\mathrm{disc}}[\mathcal{D}]$, with the following semantics. For $\mathrm{PLTL}^{\mathrm{disc}}[\mathcal{D}]$ formulas φ, ψ, a function $\eta \in \mathcal{D}$, a computation π, and an index $i \in \mathbb{N}$, we have that

—$[\![\pi^i, \mathsf{Y}\varphi]\!] = [\![\pi^{i-1}, \varphi]\!]$ if $i > 0$, and 0 otherwise.
—$[\![\pi^i, \varphi\mathsf{S}\psi]\!] = \max_{0\leq j\leq i}\{\min\{[\![\pi^j, \psi]\!], \min_{j<k\leq i}\{[\![\pi^k, \varphi]\!]\}\}\}$.
—$[\![\pi^i, \varphi\mathsf{S}_\eta\psi]\!] = \max_{0\leq j\leq i}\{\min\{\eta(i-j)[\![\pi^j, \psi]\!], \min_{j<k\leq i}\{\eta(i-k)[\![\pi^k, \varphi]\!]\}\}\}$.

Observe that, since the past is finite, then the semantics for past operators can use min and max instead of inf and sup.

As in $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$, our solution for the $\mathrm{PLTL}^{\mathrm{disc}}[\mathcal{D}]$ model-checking problem is by translating $\mathrm{PLTL}^{\mathrm{disc}}[\mathcal{D}]$ formulas to automata. The construction extends the construction for the Boolean case, which uses 2-way weak alternating automata (2AWA). The use of the obtained automata in decision procedures is similar to that in Section 4.3. In particular, it follows that the model-checking problem for $\mathrm{PLTL}^{\mathrm{disc}}[\mathcal{D}]$ with exponential discounting, namely, $\mathrm{PLTL}^{\mathrm{disc}}[E]$, is in PSPACE.

As we now show, the construction that we use when working with the "infinite future" U_η operator is similar to that of the one that we use for the "finite past" S_η operator. The key for this somewhat surprising similarity is the fact that our construction is based on a threshold. Under this threshold, we essentially bound the future that needs to be considered; thus, the fact that it is technically infinite plays no role.

A 2AWA is a tuple $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, \alpha\rangle$, where Σ, Q, q_0, α are as in AWA. The transition function is $\delta : Q \times \Sigma \to \mathcal{B}^+(\{-1, 1\} \times Q)$, that is, positive Boolean formulas over atoms of the form $\{-1, 1\} \times Q$, describing both the state to which the automaton moves and the direction in which the reading head proceeds.

As in $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$, the construction extends the construction for the Boolean case. It is not hard to extend Lemma 4.3 and generate Boolean PLTL formulas for satisfaction values in $\{0, 1\}$.

Given a $\mathrm{PLTL}^{\mathrm{disc}}[\mathcal{D}]$ formula φ and a threshold $t \in [0, 1)$, we construct a 2AWA as in Theorem 4.6 with the following additional transitions:[10]

—$\delta((\mathsf{Y}\psi > t), \sigma) = \langle -1, (\psi > t)\rangle$
—$\delta((\mathsf{Y}\psi < t), \sigma) = \langle -1, (\psi < t)\rangle$.
—$\delta((\psi_1\mathsf{S}\psi_2 > t), \sigma) = \delta((\psi_2 > t), \sigma) \vee (\delta((\psi_1 > t), \sigma) \wedge \langle -1, (\psi_1\mathsf{S}\psi_2 > t)\rangle)$.
—$\delta((\psi_1\mathsf{S}\psi_2 < t), \sigma) = \delta((\psi_2 < t), \sigma) \wedge (\delta((\psi_1 < t), \sigma) \vee \langle -1, (\psi_1\mathsf{S}\psi_2 < t)\rangle)$.

[10] In addition, the atoms in all the transitions in Theorem 4.6 are adjusted to the 2AWA syntax by replacing each atom q by the atom $\langle 1, q\rangle$.

$$
\begin{aligned}
&—\delta((\psi_1 \mathsf{S}_\eta \psi_2 > t), \sigma) = \\
&\quad \begin{cases}
\delta((\psi_2 > \frac{t}{\eta(0)}), \sigma) \vee [\delta((\psi_1 > \frac{t}{\eta(0)}), \sigma) \wedge \langle -1, (\psi_1 \mathsf{S}_{\eta^{+1}} \psi_2 > t)\rangle] & \text{if } 0 < \frac{t}{\eta(0)} < 1, \\
\texttt{False} & \text{if } \frac{t}{\eta(0)} \geq 1, \\
\delta(((\psi_1 \mathsf{S}_\eta \psi_2)^+), \sigma) & \text{if } \frac{t}{\eta(0)} = 0.
\end{cases} \\
&—\delta((\psi_1 \mathsf{S}_\eta \psi_2 < t), \sigma) = \\
&\quad \begin{cases}
\delta((\psi_2 < \frac{t}{\eta(0)}), \sigma) \wedge [\delta((\psi_1 < \frac{t}{\eta(0)}), \sigma) \vee \langle -1, (\psi_1 \mathsf{S}_{\eta^{+1}} \psi_2 < t)\rangle] & \text{if } 0 < \frac{t}{\eta(0)} \leq 1, \\
\texttt{True} & \text{if } \frac{t}{\eta(0)} > 1, \\
\texttt{False} & \text{if } \frac{t}{\eta(0)} = 0\ .
\end{cases}
\end{aligned}
$$

The correctness of the construction and the analysis of the blowup are similar to those in Section 4.2.

5.3. Changing the Tendency of Discounting

One may observe that in our discounting scheme, the value of future formulas is discounted toward 0. This, in a way, reflects an intuition that we are pessimistic about the future, or at least that we are impatient. While, in some cases, this fits the needs of the specifier, it may well be the case that we are ambivalent to the future. To capture this notion, one may want the discounting to tend to $\frac{1}{2}$. Other values are also possible. For example, it may be that we are optimistic about the future, say, when a system improves its performance while running and we know that components are likely to function better in the future. We may then want the discounting to tend, say, to $\frac{3}{4}$.

To capture this notion, we define the operator $\mathsf{O}_{\eta,z}$, parameterized by $\eta \in \mathcal{D}$ and $z \in [0, 1]$, with the following semantics.

$$
[\![\pi, \varphi \mathsf{O}_{\eta,z} \psi]\!] = \sup_{i \geq 0} \left\{ \min \left\{ \eta(i)[\![\pi^i, \psi]\!] + (1 - \eta(i))z, \min_{0 \leq j < i} \eta(j)[\![\pi^j, \varphi]\!] + (1 - \eta(j))z \right\} \right\}.
$$

The discounting function η determines the rate of convergence, and z determines the limit of the discounting. The longer it takes to fulfill the "eventuality," the closer the satisfaction value gets to z. We observe that $\varphi \mathsf{U}_\eta \psi \equiv \varphi \mathsf{O}_{\eta,0} \psi$.

Example 5.1. Consider a process scheduler. The scheduler decides which process to run at any given time. The scheduler may also run a defragment tool, but only if it is not at the expense of other processes. This can be captured by the formula $\varphi = \texttt{True}\mathsf{O}_{\eta,\frac{1}{2}}\texttt{defrag}$. Thus, the defragment tool is a "bonus": if it runs, then the satisfaction value is above $\frac{1}{2}$, but if it does not run, the satisfaction value is $\frac{1}{2}$. Treating 1 as "good" and 0 as "bad" means that $\frac{1}{2}$ is ambivalent.

We claim that Theorem 4.6 holds under the extension of $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$ with the operator O and, accordingly, so do our solutions to the search problems.

The construction of the AWA is augmented as follows. Consider a threshold $t \in [0, 1]$ and a subformula $\varphi = \psi_1 \mathsf{O}_{\eta,z} \psi_2$. We denote $\frac{t-(1-\eta(0))z}{\eta(0)}$ by τ. Recall that, in the automaton constructed in Theorem 4.6, transitions from the state, for example, $(\psi_1 \mathsf{U}_\eta \psi_2 < t)$, depend on the value of $\frac{t}{\eta(0)}$ (i.e., if this is above 1, the transition is to $\texttt{True}$). In the case of $\mathsf{O}_{\eta,z}$, we need to look at τ instead of $\frac{t}{\eta(0)}$ (as we explain later). Accordingly, the transitions from the state $(\psi_1 \mathsf{O}_{\eta,z} \psi_2 > t)$ are defined as follows.

First, if $t = z$, then $\tau = t$ and we identify the state $(\psi_1 \mathsf{O}_{\eta,z} \psi_2 > t)$ with the state $(\psi_1 \mathsf{U} \psi_2 > t)$. Otherwise, $z \neq t$ and we define:

$$—\delta((\psi_1 \mathsf{O}_{\eta,z} \psi_2 > t), \sigma) = \begin{cases} \delta((\psi_2 > \tau), \sigma) \vee \\ \quad [\delta((\psi_1 > \tau), \sigma) \wedge (\psi_1 \mathsf{O}_{\eta^{+1},z} \psi_2 > t)] & \text{if } 0 \leq \tau < 1, \\ \texttt{False} & \text{if } \tau \geq 1, \\ \texttt{True} & \text{if } \tau < 0. \end{cases}$$

$$—\delta((\psi_1 \mathsf{O}_{\eta,z} \psi_2 < t), \sigma) = \begin{cases} \delta((\psi_2 < \tau), \sigma) \wedge \\ \quad [\delta((\psi_1 < \tau), \sigma) \vee (\psi_1 \mathsf{O}_{\eta^{+1},z} \psi_2 < t)] & \text{if } 0 < \tau \leq 1, \\ \texttt{True} & \text{if } \tau > 1, \\ \texttt{False} & \text{if } \tau \leq 0. \end{cases}$$

We now proceed to show the correctness of the construction. Consider the state $(\psi_1 \mathsf{O}_{\eta,z} \psi_2 > t)$ (the dual case is similar). We claim that, for every computation π, it holds that $[\![\pi, \psi_1 \mathsf{O}_{\eta,z} \psi_2]\!] > t$ if and only if π is accepted from the state $(\psi_1 \mathsf{O}_{\eta,z} \psi_2 > t)$.

First, if $z = t$, then $[\![\pi, \psi_1 \mathsf{O}_{\eta,z} \psi_2]\!] > t$ if and only if $[\![\pi, \psi_1 \mathsf{U} \psi_2]\!] > t$ (this follows directly from the semantics of O). By the definition of these transitions, we identify the state $(\psi_1 \mathsf{O}_{\eta,z} \psi_2 > t)$ with the state $(\psi_1 \mathsf{U} \psi_2 > t)$, and correctness follows as in Theorem 4.6. Next, assume that $z \neq t$.

If $\tau < 0$, then $t < (1 - \eta(0))z$; thus, in particular, it holds that $t < \eta(0)[\![\pi^0, \psi_2]\!] + (1 - \eta(0))z$. Thus, $[\![\pi, \psi_1 \mathsf{O}_{\eta,z} \psi_2]\!] > t$; therefore, π should be accepted from $(\psi_1 \mathsf{O}_{\eta,z} \psi_2 > t)$, and the transition in this case is to `True`.

If $\tau \geq 1$, then $\eta(0) + (1 - \eta(0))z \leq t$; thus, both $\eta(0)[\![\pi^0, \psi_2]\!] + (1 - \eta(0))z \leq t$ and $\eta(0)[\![\pi^0, \psi_1]\!] + (1 - \eta(0))z \leq t$. Thus, every operand in the sup has an element less than t; therefore, the sup cannot be greater than t, and we get that $[\![\pi, \psi_1 \mathsf{O}_{\eta,z} \psi_2]\!] \leq t$. Thus, π should not be accepted from $(\psi_1 \mathsf{O}_{\eta,z} \psi_2 > t)$, and the transition in this case is to `False`.

Finally, if $0 \leq \tau < 1$, then similar to the case of U_η, we have that $[\![\pi, \psi_1 \mathsf{O}_{\eta,z} \psi_2]\!] > t$ if and only if either $[\![\pi, \psi_2]\!] > \tau$ or both $[\![\pi, \psi_1]\!] > \tau$ and $[\![\pi^1, \psi_1 \mathsf{O}_{\eta^{+1},z} \psi_2]\!] > t$. This condition is captured by the corresponding transition from $(\psi_1 \mathsf{O}_{\eta,z} \psi_2 > t)$.

It remains to show that there are only finitely many reachable states from the initial state. Since $z \neq t$, we get that $\lim_{\eta(0) \to 0} \tau = \{^{\infty}_{-\infty} \; ^{t > z}_{t < z}$. Thus, after a finite number of transitions, τ takes a value that is not in $[0, 1]$, in which case, the next state is `True` or `False`; thus, the number of states reachable from $\varphi > t$ is finite.

We remark that the latter observation is not true if $z = t$, which is why we needed to handle this case separately.

5.4. Approximating LTL$^{\text{disc}}[\mathcal{D}]$ Problems

From a practical point of view, a designer may not care about the exact satisfaction value of a formula, and would settle for an approximate value. To be precise, let $\epsilon > 0$ and consider a discounting function η. We define a new function $\eta|_{>\epsilon}$ by $\eta|_{>\epsilon}(n) = \eta(n)$ for all n such that $\eta(n) > \epsilon$, and $\eta|_{>\epsilon} = 0$ otherwise. Note that $\eta|_{>\epsilon}$ is not strictly decreasing, therefore is not a legal discounting function. Still, the semantics of LTL$^{\text{disc}}[\mathcal{D}]$ with these type of discounting functions is well defined. Moreover, consider an LTL$^{\text{disc}}[\mathcal{D}]$ formula φ, and let $\epsilon > 0$. We obtain from φ the formula $\varphi|_{>\epsilon}$ by replacing every discounting function η with $\eta|_{>\epsilon}$. It is easy to prove by induction over the structure of φ that, for every computation π, it holds that $|[\![\pi, \varphi]\!] - [\![\pi, \varphi|_{>\epsilon}]\!]| < \epsilon^{|\varphi|}$. Thus, if we are only interested in the satisfaction value of φ up to some $\delta > 0$, it is enough to reason about $\varphi|_{>\epsilon}$ for some ϵ such that $\epsilon^{|\varphi|} < \delta$.

Finally, observe that every operator of the form $U_{\eta|_{>\epsilon}}$ can be described by a propositional quality operator—the value of $[\![\pi, \psi_1 U_{\eta|_{>\epsilon}} \psi_2]\!]$ is determined after a finite prefix of π, whose length is bound by the first index n for which $\eta(n) \leq \epsilon$.

Thus, $\varphi|_{>\epsilon}$ can actually be described as an LTL$[\mathcal{F}]$ formula, on which we can reason exactly both for model-checking and synthesis purposes.

Another approach to approximation of $\text{LTL}^{\text{disc}}[\mathcal{D}]$ was taken in Nakagawa and Hasuo [2015], in which the goal is to find a path in a Kripke structure that satisfies a formula with almost-optimal value.

6. COMBINING TEMPORAL AND PROPOSITIONAL QUALITY

As model checking is decidable for $\text{LTL}^{\text{disc}}[\mathcal{D}]$, one may wish to push the limit and extend the expressive power of the logic. In particular, of great interest is the combination of discounting with propositional quality operators.

In this section, we examine such combinations. Interestingly, as it turns out, adding certain propositional quality operators (i.e., weighted average) renders the model-checking problem undecidable, while other, simpler operators (i.e., unary multiplication) do not add expressive power, and do not change the decidability status of the logic.

6.1. Adding the Average Operator

A well-motivated extension is the introduction of the average operator $\oplus$, with the semantics $[\![\pi, \varphi \oplus \psi]\!] = \frac{[\![\pi,\varphi]\!]+[\![\pi,\psi]\!]}{2}$. As we have seen in Section 2, this operator is useful to express the affect of different components on the overall quality of the system (see Example 2.1).

We show that adding the $\oplus$ operator to $\text{LTL}^{\text{disc}}[\mathcal{D}]$ gives a logic, denoted $\text{LTL}^{\text{disc}\oplus}[\mathcal{D}]$, for which the satisfiability and model-checking problems are undecidable, both in their strict and nonstrict versions. That is, for every $\sim \in \{<, \leq, =, \geq, >\}$, it is undecidable, given a formula φ, a system $\mathcal{K}$, and a threshold v, whether $[\![K, \varphi]\!] \sim v$, and whether there exists a computation π such that $[\![\pi, \varphi]\!] \sim v$.

The proofs are arranged in the following structure. We start by showing that the validity problem for $\text{LTL}^{\text{disc}\oplus}[\mathcal{D}]$ is undecidable, then extract ideas from the proof, which are later used to show that the rest of the problems are undecidable. Our proofs apply to $\text{LTL}^{\text{disc}\oplus}[\mathcal{D}]$ with every nonempty set of discounting functions $\mathcal{D}$.

The validity problem asks, given an $\text{LTL}^{\text{disc}\oplus}[\mathcal{D}]$ formula φ over the atomic propositions AP and a threshold $v \in [0, 1]$, whether $[\![\pi, \varphi]\!] > v$ for every $\pi \in (2^{AP})^{\omega}$.

In the first undecidability proof, we show a reduction from the 0-halting problem for two-counter machines. A *two-counter machine* $\mathcal{M}$ is a sequence $(l_1, \ldots, l_n)$ of commands involving two counters x and y. We refer to $\{1, \ldots, n\}$ as the *locations* of the machine. There are five possible forms of commands:

$$\text{INC}(c),\ \text{DEC}(c),\ \text{GOTO } l_i,\ \text{IF } c{=}0 \text{ GOTO } l_i \text{ ELSE GOTO } l_j,\ \text{HALT},$$

where $c \in \{x, y\}$ is a counter and $1 \leq i, j \leq n$ are locations. A *halting run* of a two-counter machine $\mathcal{M}$ is a sequence $\rho = \rho_1, \ldots, \rho_m \in (L \times \mathbb{N} \times \mathbb{N})^*$ such that the following hold:

(1) $\rho_1 = \langle l_1, 0, 0\rangle$.
(2) For all $1 < i \leq m$, let $\rho_{i-1} = (l_k, \alpha, \beta)$ and $\rho_i = (l', \alpha', \beta')$. Then, the following hold:
   —If l_k is an $\text{INC}(x)$ command (resp., $\text{INC}(y)$), then $\alpha' = \alpha + 1$, $\beta' = \beta$ (resp., $\beta' = \beta + 1$, $\alpha' = \alpha$), and $l' = l_{k+1}$.
   —If l_k is a $\text{DEC}(x)$ command (resp., $\text{DEC}(y)$), then $\alpha' = \alpha - 1$, $\beta' = \beta$ (resp., $\beta' = \beta - 1$, $\alpha' = \alpha$), and $l' = l_{k+1}$.
   —If l_k is a GOTO l_s command, then $\alpha' = \alpha$, $\beta' = \beta$, and $l' = l_s$.
   —If l_k is an IF x=0 GOTO l_s ELSE GOTO l_t command, then $\alpha' = \alpha$, $\beta' = \beta$, and $l' = l_s$ if $\alpha = 0$, and $l' = l_t$ otherwise.

—If l_k is an IF y=0 GOTO l_s ELSE GOTO l_t command, then $\alpha' = \alpha$, $\beta' = \beta$, and $l' = l_s$ if $\beta = 0$, and $l' = l_t$ otherwise.
—If l' is a HALT command, then $i = m$. That is, a run does not continue after HALT.

(3) $\rho_m = \langle l_k, \alpha, \beta \rangle$ such that l_k is a HALT command.

Observe that the machine $\mathcal{M}$ is deterministic. We say that $\mathcal{M}$ 0-halts if it halts with both counters having value 0. That is, $\mathcal{M}$ 0-halts if there exists $l \in L$ that is a HALT command, such that the run of $\mathcal{M}$ ends in $\langle l, 0, 0 \rangle$.

We say that a sequence of commands $\tau \in L^*$ *fits* a run ρ if τ is the projection of ρ on its first component.

Since we can always check whether $c = 0$ before a DEC(c) command, we assume that the machine never executes DEC(c) with $c = 0$. That is, the counters never have negative values. Given a counter machine $\mathcal{M}$, deciding whether $\mathcal{M}$ halts is known to be undecidable [Minsky 1967]. Given $\mathcal{M}$, deciding whether $\mathcal{M}$ 0-halts, termed the 0-*halting problem*, is also undecidable: given a counter machine $\mathcal{M}$, we can replace every HALT command with a code that clears the counters before halting. In fact, from this, we see that the *promise problem*, namely, the problem of deciding whether $\mathcal{M}$ 0-halts given the promise that either it 0-halts or it does not halt at all, is also undecidable.

THEOREM 6.1. *The validity problem for* $\mathrm{LTL}^{disc\oplus}[\mathcal{D}]$ *is undecidable for every* $\mathcal{D} \neq \emptyset$.

PROOF. We start by showing a reduction from the 0-halting problem for two-counter machines to the following problem: given an $\mathrm{LTL}^{\mathrm{disc}\oplus}[\mathcal{D}]$ formula φ over the atomic propositions AP, whether there exists a computation $\pi \in AP^\omega$ such that $[\![\pi, \varphi]\!] \geq \frac{1}{2}$. We dub this the $\frac{1}{2}$-*co-validity* problem. We will later reduce this problem to the (complement of the) validity problem.

We construct from $\mathcal{M}$ an $\mathrm{LTL}^{\mathrm{disc}\oplus}[\mathcal{D}]$ formula φ such that $\mathcal{M}$ 0-halts if and only if there exists a computation π such that $[\![\pi, \varphi]\!] \geq \frac{1}{2}$. The idea behind the construction is as follows. The formula φ is interpreted over computations over the atomic propositions $AP = \{1, \ldots, n, \#, x, y\}$, where $1, \ldots, n$ are the commands of M. The computation over which φ is interpreted corresponds to a description of a run of $\mathcal{M}$, where every triplet $\langle l_i, \alpha, \beta \rangle$ is encoded as the string $ix^\alpha y^\beta \#$. We ensure that computations that satisfy φ with value greater than 0 are such that, in every position, only a single atomic proposition is true.

Example 6.2. Consider the following machine $\mathcal{M}$:

l_1: INC(x)
l_2: IF x=0 GOTO l_6 ELSE GOTO l_3
l_3: INC(y)
l_4: DEC(x)
l_5: GOTO (l_2)
l_6: DEC(y)
l_7: HALT

The command sequence that represents the run of this machine is

$$\langle l_1, 0, 0 \rangle, \langle l_2, 1, 0 \rangle, \langle l_3, 1, 0 \rangle, \langle l_4, 1, 1 \rangle, \langle l_5, 0, 1 \rangle, \langle l_2, 0, 1 \rangle, \langle l_6, 0, 1 \rangle, \langle l_7, 0, 0 \rangle$$

and the encoding of it as a computation is

$$1\#2x\#3x\#4xy\#5y\#2y\#6y\#7\#.$$

The formula φ "states" (recall that the setting is quantitative, not Boolean) the following properties of the computation π:

(1) The first configuration in π is the initial configuration of $\mathcal{M}$ ($\langle l_1, 0, 0\rangle$, or 1# in our encoding).
(2) The last configuration in π is $\langle l, 0, 0\rangle$ (or k in our encoding), for which l can be any line whose command is HALT.
(3) π represents a legal run of $\mathcal{M}$, up to the consistency of the counters between transitions.
(4) The counters are updated correctly between configurations.

As we show later, properties 1 to 3 can easily be specified by LTL formulas, such that computations that satisfy properties 1 to 3 get satisfaction value 1. Property 4 utilizes the expressive power of $\text{LTL}^{\text{disc}\oplus}[\mathcal{D}]$, as we now demonstrate. The intuition behind property 4 is the following. We need to compare the value of a counter before and after a command, such that the formula takes a low value if a violation is encountered, and a high value otherwise. Specifically, the formula that we construct takes value $\frac{1}{2}$ if no violation occurred and a lower value if a violation did occur.

We start with a simpler case to demonstrate the point. Let $\eta \in \mathcal{D}$ be a discounting function. Consider the formula $CountA := a\mathsf{U}_\eta \neg a$ and the computation $a^i b^j \#^\omega$. It holds that $[\![a^i b^j, CountA]\!] = \eta(i)$. Similarly, it holds that $[\![a^i b^j \#^\omega, a\mathsf{U}(b\mathsf{U}_\eta \neg b)]\!] = \eta(j)$. Denote the latter by $CountB$. Let

$$CompareAB := (CountA \oplus \neg CountB) \wedge (\neg CountA \oplus CountB).$$

We now have that

$$[\![a^i b^j \#^\omega, CompareAB]\!] = \min\left\{\frac{\eta(i)+1-\eta(j)}{2}, \frac{\eta(j)+1-\eta(i)}{2}\right\} = \frac{1}{2} - \frac{|\eta(i)-\eta(j)|}{2},$$

and observe that the latter is $\frac{1}{2}$ if and only if $i = j$, and is less than $\frac{1}{2}$ otherwise. This is because η is strictly decreasing and, in particular, an injection.

Thus, we can compare counters. To apply this technique to the encoding of a computation, we only need some technical formulas to "parse" the input and find consecutive occurrences of a counter.

We now dive into the technical definition of φ. The atomic propositions are $AP = \{1, \ldots, n, \#, x, y\}$ (where $l_1, \ldots, l_n$ are the commands of $\mathcal{M}$). We let $\varphi := CheckCmds \wedge CheckInit \wedge CheckFinal \wedge CheckCounters \wedge ForceSingletons$.

ForceSingletons. This formula ensures that for a computation to get a value of more than 0, every letter in the computation must be a singleton. Formally,

$$ForceSingletons := \mathsf{G}\left(\bigvee_{p\in AP}\left(p \wedge \bigwedge_{q\in AP\setminus\{p\}} \neg q\right)\right).$$

CheckInit and CheckFinal. These formulas check that the initial and final configurations are correct, that after the final configuration there are only #s, and that the final configuration is reached eventually.

$$CheckInit := 1 \wedge \mathsf{X}\#.$$

Let $I = \{i : l_i = \text{HALT}\}$; we define

$$CheckFinal := \mathsf{G}\left(\left(\bigvee_{i\in I} i\right) \rightarrow \mathsf{XG}\#\right) \wedge \left(\mathsf{F}\left(\bigvee_{i\in I} i\right)\right).$$

Note that $CheckFinal$ also ensures that the counters are 0.

CheckCmds. This formula verifies that the local transitions follow the instructions in the counter machine, ignoring the consistency of the counter values, but enforcing that a jump behaves according to the counters. We start by defining, for every $i \in \{1, \ldots, n\}$, the formula:

$$waitfor(i) := (x \vee y)\mathsf{U}(\# \wedge \mathsf{X}i).$$

Intuitively, a computation satisfies this formula (i.e., gets value 1) if and only if it reads counter descriptions until the next delimiter, and the next command is l_i.

Now, for every $i \in \{1, \ldots, n\}$, we define ψ_i as follows.

—If $l_i =$ GOTO l_j, then $\psi_i := \mathsf{X}waitfor(j)$.
—If $l_i \in \{\text{INC}(c), \text{DEC}(c) : c \in \{x, y\}\}$, then $\psi_i := \mathsf{X}waitfor(i+1)$. [11]
—If $l_i =$ IF x=0 GOTO l_j ELSE GOTO l_k, then $\psi_i := \mathsf{X}((x \rightarrow waitfor(k)) \wedge ((\neg x) \rightarrow waitfor(j)))$.
—If $l_i =$ IF y=0 GOTO l_j ELSE GOTO l_k, then $\psi_i := \mathsf{X}((x\mathsf{U}y) \wedge waitfor(k)) \vee ((x\mathsf{U}\#) \wedge waitfor(j)))$.
—If $l_i =$ HALT we do not really need additional constraints, due to $CheckFinal$. Thus, we have that $\psi_i =$ `True`.

Finally, we define $CheckCmds := \mathsf{G} \bigwedge_{i \in \{1,\ldots,n\}} (i \rightarrow \psi_i)$.

CheckCounters. This is the heart of the construction. The formula checks whether consecutive occurrences of the counters match the transition between the commands. We start by defining $countX := x\mathsf{U}_\eta \neg x$ and $countY := x\mathsf{U}(y\mathsf{U}_\eta \neg y)$. Similarly, we have that $countX^{-1} = x\mathsf{U}_\eta \mathsf{X} \neg x$ and $countY^{-1} = x\mathsf{U}(y\mathsf{U}_\eta \mathsf{X} \neg y)$.

We need to define a formula to handle some edge cases.

Let $I_{\text{HALT}} = \{i : l_i = \text{HALT}\}$, and similarly define $I_{\text{DEC}(x)}$ and $I_{\text{DEC}(y)}$. We define

$$\begin{aligned} Last := & \bigvee_{i \in I_{\text{DEC}(x)}} i \wedge \mathsf{X}\left(x \wedge \mathsf{X}\left(\# \wedge \mathsf{X} \bigvee_{k \in I_{\text{HALT}}} k\right)\right) \vee \\ & \bigvee_{i \in I_{\text{DEC}(y)}} i \wedge \mathsf{X}\left(y \wedge \mathsf{X}\left(\# \wedge \mathsf{X} \bigvee_{k \in I_{\text{HALT}}} k\right)\right) \vee \\ & \bigvee_{i \in \{1,\ldots,n\}} i \wedge \mathsf{X}\left(\# \wedge \mathsf{X} \bigvee_{k \in I_{\text{HALT}}} k\right). \end{aligned}$$

Intuitively, the formula $Last$ holds exactly in the last transition, that is, before the final 0-halting configuration.

Testing the counters involves six types of comparisons: checking equality, increasing by 1, and decreasing by 1 for each of the two counters. We define the following formulas for these tests. To explain the formulas, consider, for example, the formula $Comp(x, =)$. This formula compares the number of xs in the current configuration, with the number of xs in the next configuration. The comparison is based on the comparison that we explained earlier, and is augmented by some parsing, as we need to reach the next configuration before comparing.

—$Comp(x, =) := (countX \oplus (x\mathsf{U}(y\mathsf{U}(\# \wedge \mathsf{XX}\neg countX))))$
$\wedge ((\neg countX) \oplus (x\mathsf{U}(y\mathsf{U}(\# \wedge \mathsf{XX}countX))))$.

[11] If $i = n$, then this line can be omitted from the initial machine, so that without loss of generality this does not happen.

—$Comp(y, =) := (countY \oplus (x\mathsf{U}(y\mathsf{U}(\# \wedge \mathsf{XX}\neg countY))))$
$\wedge((\neg countY) \oplus (x\mathsf{U}(y\mathsf{U}(\# \wedge \mathsf{XX}countY))))$.
—$Comp(x, +1) := (countX \oplus (x\mathsf{U}(y\mathsf{U}(\# \wedge \mathsf{XX}\neg countX^{-1}))))$
$\wedge((\neg countX) \oplus (x\mathsf{U}(y\mathsf{U}(\# \wedge \mathsf{XX}countX^{-1}))))$.
—$Comp(y, +1) := (countY \oplus (x\mathsf{U}(y\mathsf{U}(\# \wedge \mathsf{XX}\neg countY^{-1}))))$
$\wedge((\neg countY) \oplus (x\mathsf{U}(y\mathsf{U}(\# \wedge \mathsf{XX}countY^{-1}))))$.
—$Comp(x, -1) := (countX^{-1} \oplus (x\mathsf{U}(y\mathsf{U}(\# \wedge \mathsf{XX}\neg countX))))$
$\wedge((\neg countX^{-1}) \oplus (x\mathsf{U}(y\mathsf{U}(\# \wedge \mathsf{XX}countX))))$.
—$Comp(y, -1) := (countY^{-1} \oplus (x\mathsf{U}(y\mathsf{U}(\# \wedge \mathsf{XX}\neg countY))))$
$\wedge((\neg countY^{-1}) \oplus (x\mathsf{U}(y\mathsf{U}(\# \wedge \mathsf{XX}countY))))$.

Now, for every $i \in \{1, \ldots, n\}$, we define ξ_i as follows.

—If $l_i \in \{\text{GOTO } l_j, \text{ IF } c\text{=0 GOTO } l_j \text{ ELSE GOTO } l_k : c \in \{x, y\}\}$, then we need to make sure that the values of the counters do not change. Accordingly, we define $\xi_i := (X(comp(x, =) \wedge comp(y, =)) \vee Last$.
—If $l_i = \text{INC}(x)$, then we need to make sure that x increases and y does not change. Accordingly, we define $\xi_i := (X(comp(x, +1) \wedge comp(y, =)) \vee Last$.
—If $l_i = \text{INC}(y)$, then $\xi_i := (X(comp(x, =) \wedge comp(y, +1)) \vee Last$.
—If $l_i = \text{DEC}(x)$, then $\xi_i := (X(comp(x, -1) \wedge comp(y, =)) \vee Last$.
—If $l_i = \text{DEC}(y)$, then $\xi_i := (X(comp(x, =) \wedge comp(y, -1)) \vee Last$.
—If $l_i = \text{HALT}$, then we do not need additional constraints, due to $CheckFinal$. Accordingly, we define $\xi_i = \texttt{True}$.

Finally, we define $CheckCounters := \mathsf{G} \bigwedge_{i \in \{1,\ldots,n\}} (i \rightarrow \xi_i)$.

In order to establish the correctness of the construction, we first observe that, if $\mathcal{M}$ 0-halts, then the description of its run is a computation π such that $[\![\pi, \varphi]\!] = \frac{1}{2}$. The computation satisfies the formulas $CheckCmds$, $CheckInit$, $CheckFinal$, and $ForceSingletons$ with value 1, and the formula $CheckCounters$ with value $\frac{1}{2}$. Conversely, consider a computation π such that $[\![\pi, \varphi]\!] = \frac{1}{2}$. Since $CheckCmds$, $CheckInit$, $CheckFinal$, and $ForceSingletons$ are all Boolean LTL formulas, we get that π satisfies each of them with value 1 (otherwise, we would have $[\![\pi, \varphi]\!] = 0$). It is easy to verify that these formulas enforce π to describe a 0-halting computation of $\mathcal{M}$, which is legal up to counter updates. Finally, it must hold that $[\![\pi, CheckCounters]\!] \geq \frac{1}{2}$, and by the structure of $CheckCoutners$ we, in fact, have $[\![\pi, CheckCounters]\!] = \frac{1}{2}$, which implies that the counter-updates behave according to the commands. We conclude that π represents a legal 0-halting computation of $\mathcal{M}$, thus $\mathcal{M}$ 0-halts.

Finally, we reduce the $\frac{1}{2}$-co-validity problem to the complement of the validity problem: given a formula φ, the reduction outputs $\langle \neg\varphi, \frac{1}{2} \rangle$. Now, there exists a computation π such that $[\![\pi, \varphi]\!] \geq \frac{1}{2}$ if and only if there exists a computation π such that $[\![\pi, \neg\varphi]\!] \leq \frac{1}{2}$ if and only if it is not true that $[\![\pi, \neg\varphi]\!] > \frac{1}{2}$ for every computation π. Thus, φ is $\frac{1}{2}$-co-valid if and only if $\neg\varphi$ is not valid for threshold $\frac{1}{2}$. We conclude that the validity problem is undecidable. □

The *model-checking* problem (resp., *strict* model-checking problem) is to decide, given a Kripke structure $\mathcal{K}$, a formula φ, and a threshold v, whether $[\![K, \varphi]\!] \geq v$ (resp., $[\![\mathcal{K}, \varphi]\!] > v$). We now show that both variants of the model-checking problem are undecidable for $\text{LTL}^{\text{disc}\oplus}[\mathcal{D}]$. For this, we first pinpoint the essential technical details in the reduction in the proof of Theorem 6.1. The following are properties of the formula $\psi = \neg\varphi$, where φ is the formula constructed in the proof of Theorem 6.1.

LEMMA 6.3. *Given a two-counter machine $\mathcal{M}$ that is promised to either* 0*-halt or not to halt at all, there exists, for every $\mathcal{D} \neq \emptyset$, an* $\mathrm{LTL}^{disc\oplus}[\mathcal{D}]$ *formula ψ such that, for every computation π that represents a computation of $\mathcal{M}$, the following hold:*

(1) *If π is a legal halting computation of $\mathcal{M}$, then $[\![\pi, \psi]\!] = \frac{1}{2}$.*
(2) *If π cheats in a transition between commands, then $[\![\pi, \psi]\!] = 1$.*
(3) *If π cheats in the counter values, then $[\![\pi, \psi]\!] = \frac{1}{2} + \epsilon$ such that $\epsilon \geq \frac{1}{2}(\eta(i) - \eta(i+1))$ for the maximal difference $\eta(i) - \eta(i+1)$, where i is a counter value in π.*

Before turning to the proofs, let us briefly explain why these results are nontrivial. For the nonstrict model-checking problem, there does not seem to be an immediate reduction from the validity problem. In the proof of Theorem 6.1, we have that $[\![\mathcal{K}, \varphi]\!] = \frac{1}{2}$ always (for the system $\mathcal{K}$ that generates every computation).

For the strict model-checking problem, it is tempting to say that $[\![\mathcal{K}, \varphi]\!] > v$ if and only if for every computation π of $\mathcal{K}$, it holds that $[\![\pi, \varphi]\!] > v$. However, this is incorrect, since it may be the case that $[\![\pi, \varphi]\!] > v$ for every computation π, yet these values can get arbitrarily close to v, then $[\![\mathcal{K}, \varphi]\!] = v$. This convergence of the satisfaction value is at the heart of the problem; avoiding it is the crux in our proofs.

We now turn to show that the $\mathrm{LTL}^{\mathrm{disc}\oplus}[\mathcal{D}]$ model-checking and strict model-checking problems are undecidable. The proofs apply to every nonempty set of discounting functions $\mathcal{D}$.

THEOREM 6.4. *The strict model-checking problem for* $\mathrm{LTL}^{disc\oplus}[\mathcal{D}]$ *is undecidable for every $\mathcal{D} \neq \emptyset$.*

PROOF. Assume, by way of contradiction, that the strict model-checking problem is decidable. We show how to decide the 0-halting promise problem for two-counter machines, thus reaching a contradiction.

Given a two-counter machine $\mathcal{M}$ that is promised to either 0-halt or not halt at all, construct the formula φ as per Lemma 6.3, and consider the Kripke structure $\mathcal{K}$ that generates every computation. Observe that, by Lemma 6.3, it holds that $[\![\mathcal{K}, \varphi]\!] \geq \frac{1}{2}$.

Decide whether $[\![\mathcal{K}, \varphi]\!] > \frac{1}{2}$. If $[\![\mathcal{K}, \varphi]\!] > \frac{1}{2}$, then, for every computation π, it holds that $[\![\pi, \varphi]\!] > \frac{1}{2}$, and by Lemma 6.3 we conclude that $\mathcal{M}$ does not halt.

If $[\![\mathcal{K}, \varphi]\!] \leq \frac{1}{2}$, then $[\![\mathcal{K}, \varphi]\!] = \frac{1}{2}$. We observe that there are now two possible cases:

(1) $\mathcal{M}$ halts.
(2) $\mathcal{M}$ does not halt, and for every n, there are computations that reach HALT while cheating in counter values larger than n, and not cheating in the commands.

We show how to distinguish between cases 1 and 2.

Consider the $\mathrm{LTL}^{\mathrm{disc}\oplus}[\mathcal{D}]$ formula

$$\psi = \mathtt{True} \oplus (\mathsf{G}(x \mathsf{U}_\eta \neg x) \wedge \mathsf{G}(y \mathsf{U}_\eta \neg y)).$$

It is not hard to verify that, for every computation π that represents a computation of $\mathcal{M}$, it holds that $[\![\pi, \psi]\!] = \frac{1}{2} + \frac{1}{2}\epsilon$, where $\epsilon = \inf(\eta(i) : i$ is the value of a counter in $\pi)$. Let $\xi = \varphi \vee \psi$.

If $\mathcal{M}$ halts (case 1), then for every computation π, we have one of the following.

a. π describes a legal halting run of $\mathcal{M}$, in which case $[\![\pi, \varphi]\!] = \frac{1}{2}$ and $[\![\pi, \psi]\!] > \frac{1}{2} + \epsilon$ for some $\epsilon > 0$ (independent of π), since the counters are bounded. Thus, $[\![\pi, \xi]\!] > \frac{1}{2} + \epsilon$.
b. π cheats in the commands, in which case $[\![\pi, \varphi]\!] = 1$; thus, $[\![\pi, \xi]\!] = 1$.

c. π cheats in the counters, in which case, since the counters are bounded, the first cheat must occur with small counters. Thus, $[\![\pi, \varphi]\!] > \frac{1}{2} + \epsilon$ for some $\epsilon > 0$ independent of π. Thus, $[\![\pi, \xi]\!] > \frac{1}{2} + \epsilon$.

In all three cases, we get that $[\![\pi, \xi]\!] > \frac{1}{2} + \epsilon$; thus, $[\![\mathcal{K}, \xi]\!] > \frac{1}{2}$.

If $\mathcal{M}$ does not halt (case 2), then, for every n and for every computation π that cheats with counters larger than n, it holds that $[\![\pi, \varphi]\!] < \frac{1}{2} + \epsilon$, where $\epsilon = \epsilon(n) \to 0$ as $n \to \infty$. Since the counters in π are large, it also holds that $[\![\pi, \psi]\!] < \frac{1}{2} + \epsilon'$, where $\epsilon' = \epsilon'(n) \to 0$ as $n \to \infty$. We conclude that there exists a sequence of computations whose satisfaction values in ξ tend to $\frac{1}{2}$; thus, $[\![\mathcal{K}, \xi]\!] = \frac{1}{2}$.

Thus, in order to distinguish between cases 1 and 2, it is enough to decide whether $[\![\mathcal{K}, \xi]\!] > \frac{1}{2}$.

To conclude, the algorithm for deciding whether $\mathcal{M}$ 0-halts is as follows. Start by constructing φ. If $[\![\mathcal{K}, \varphi]\!] > \frac{1}{2}$, then $\mathcal{M}$ does not halt. Otherwise, construct ξ. If $[\![\mathcal{K}, \xi]\!] > \frac{1}{2}$, then $\mathcal{M}$ halts, and otherwise $\mathcal{M}$ does not halt. □

THEOREM 6.5. *The model-checking problem for* $\text{LTL}^{disc\oplus}[\mathcal{D}]$ *is undecidable for every* $\mathcal{D} \neq \emptyset$.

PROOF. Recall that, for every Kripke structure $\mathcal{K}$ and formula φ, it holds that $[\![\mathcal{K}, \varphi]\!] \geq v$ if and only if there does not exist a computation π of $\mathcal{K}$ such that $[\![\pi, \neg\varphi]\!] > 1 - v$.

We show that the latter problem is undecidable even if we fix $\mathcal{K}$ to be the system that generates every computation.

We show a reduction from the 0-halting promise problem to the latter problem. Given a two-counter machine $\mathcal{M}$ that is promised to either 0-halt or not halt at all, construct the formula φ as per Lemma 6.3 and the formula ψ such that, for every computation π, we have that $[\![\pi, \psi]\!] = \frac{1}{2} + \frac{1}{2}\epsilon$, where $\epsilon = \inf(\eta(i) - \eta(i+1) : i$ is the value of a counter in $\pi)$. The formula ψ can be defined as

$$\psi = \mathsf{G}((x\mathsf{U}_\eta \neg x) \oplus \neg((x \vee \mathsf{X}x)\mathsf{U}_\eta(\neg x \wedge \neg\mathsf{X}x)) \wedge (y\mathsf{U}_\eta \neg y) \oplus \neg((y \vee \mathsf{X}y)\mathsf{U}_\eta(\neg y \wedge \neg\mathsf{X}y))).$$

Let $\theta = (\neg\varphi) \oplus \psi$. We claim that $\mathcal{M}$ halts if and only if there exists a computation π such that $[\![\pi, \theta]\!] > \frac{1}{2}$.

If $\mathcal{M}$ halts, then, for the computation π that describes the halting run of $\mathcal{M}$, it holds that $[\![\pi, \varphi]\!] = \frac{1}{2}$; thus, $[\![\pi, \neg\varphi]\!] = \frac{1}{2}$. Since the counters in π are bounded (as the run is halting), then $[\![\pi, \psi]\!] > \frac{1}{2}$; thus, $[\![\pi, \theta]\!] > \frac{1}{2}$.

If $\mathcal{M}$ does not halt, consider a computation π.

—If π cheats in the commands, then $[\![\pi, \neg\varphi]\!] = 0$; thus, $[\![\pi, \theta]\!] = 0 + \frac{1}{2}[\![\pi, \psi]\!] \leq \frac{1}{2}$.
—If π cheats in the counters, then $[\![\pi, \neg\varphi]\!] = \frac{1}{2} - \frac{1}{2}\epsilon$ and $[\![\pi, \psi]\!] = \frac{1}{2} + \frac{1}{2}\epsilon'$, where $\epsilon \geq \frac{1}{2}(\eta(i) - \eta(i+1)) = \epsilon'$ for the smallest difference $\eta(i) - \eta(i+1)$ in π. Thus, $[\![\pi, \theta]\!] \leq \frac{1}{2}$. □

Finally, using simple reductions, we can obtain the following.

THEOREM 6.6. *For every* $\sim \in \{<, \leq, =, \geq, >\}$ *and for every* $\mathcal{D} \neq \emptyset$, *the following problems are undecidable.*

—*Model checking: Given an* $\text{LTL}^{disc\oplus}[\mathcal{D}]$ *formula* φ, *a system* $\mathcal{K}$, *and a threshold* v, *whether* $[\![K, \varphi]\!] \sim v$.
—*Satisfiability: Given an* $\text{LTL}^{disc\oplus}[\mathcal{D}]$ *formula* φ, *a system* $\mathcal{K}$, *and a threshold* v, *whether there exists a computation* π *such that* $[\![\pi, \varphi]\!] \sim v$.

—*Validity: Given an* $\mathrm{LTL}^{disc\oplus}[\mathcal{D}]$ *formula* φ*, a system* $\mathcal{K}$*, and a threshold* v*, whether* $[\![\pi, \varphi]\!] \sim v$ *for every computation* π.

Proof. We state the main ideas, leaving the technical details to the reader. All the results are simple reductions, using Theorems 6.4, 6.5, and the following observations:

(1) From Theorem 6.1, we get that the validity problem for the "$>$" case is undecidable.
(2) Theorem 6.1 also shows that the satisfiability problem for the "$=$" case is undecidable.
(3) The proofs of Theorems 6.1, 6.4, and 6.5 use a Kripke structure that generates every computation.

To demonstrate the ideas in the proof, we show the undecidability of some of the cases.

—The model-checking problem for the "$<$" case is undecidable, since it is the complement of the model checking for the "$\geq$" case.
—The satisfiability problem for the $<$ case is undecidable, since, by Observation 3, it is equivalent to the model checking for the "$<$" case.
—The satisfiability problem for the $>$ case is undecidable, since it amounts to deciding the "$<$" case of satisfiability for the formula $\neg\varphi$ and the threshold $1 - v$.

Similar short arguments prove the undecidability of the other cases. □

6.2. Combining $\mathbf{LTL}^{\mathbf{disc}}[\mathcal{D}]$ and $\mathbf{LTL}^{\triangledown}$

As shown in Section 6.1, adding the operator $\oplus$ to $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$ makes model checking undecidable. One may still want to find propositional quality operators that we can add to the logic retaining its decidability. We show later that the logic $\mathrm{LTL}^{\triangledown}$ (see Section 3.4) consists of such propositional quality operators.

As elaborated in Section 3.4, $\mathrm{LTL}^{\triangledown}$ is a fragment of $\mathrm{LTL}[\mathcal{F}]$ that covers the possible linear approaches of formalizing quality as a value between true and false. It contains the standard Boolean operators and three propositional quality operators – $\triangledown_\lambda$, $\blacktriangledown_\lambda$, and $\blacktriangledown\!\!\blacktriangledown_\lambda$—aimed at capturing the *competence*, *necessity*, and *confidence*, respectively, of the specifications.

We handle these operators by adding the following transitions to the construction in the proof of Theorem 4.6.

$$\text{—}\delta(\triangledown_\lambda\varphi > t, \sigma) = \begin{cases} \delta(\varphi > \frac{t}{\lambda}, \sigma) & \text{if } \frac{t}{\lambda} < 1, \\ \texttt{False} & \text{if } \frac{t}{\lambda} \geq 1, \end{cases}$$

$$\text{—}\delta(\triangledown_\lambda\varphi < t, \sigma) = \begin{cases} \delta(\varphi < \frac{t}{\lambda}, \sigma) & \text{if } \frac{t}{\lambda} \leq 1, \\ \texttt{True} & \text{if } \frac{t}{\lambda} > 1. \end{cases}$$

$$\text{—}\delta(\blacktriangledown_\lambda\varphi > t, \sigma) = \begin{cases} \delta(\varphi > \frac{t+\lambda-1}{\lambda}, \sigma) & \text{if } \frac{t+\lambda-1}{\lambda} < 1, \\ \texttt{False} & \text{if } \frac{t+\lambda-1}{\lambda} \geq 1, \end{cases}$$

$$\text{—}\delta(\blacktriangledown_\lambda\varphi < t, \sigma) = \begin{cases} \delta(\varphi < \frac{t+\lambda-1}{\lambda}, \sigma) & \text{if } \frac{t+\lambda-1}{\lambda} \leq 1, \\ \texttt{True} & \text{if } \frac{t+\lambda-1}{\lambda} > 1. \end{cases}$$

$$\text{—}\delta(\blacktriangledown\!\!\blacktriangledown_\lambda\varphi > t, \sigma) = \begin{cases} \delta(\varphi > \frac{2t+\lambda-1}{2\lambda}, \sigma) & \text{if } \frac{t}{\lambda} < 1, \\ \texttt{False} & \text{if } \frac{2t+\lambda-1}{2\lambda} \geq 1, \end{cases}$$

$$\text{—}\delta(\blacktriangledown\!\!\blacktriangledown_\lambda\varphi < t, \sigma) = \begin{cases} \delta(\varphi < \frac{2t+\lambda-1}{2\lambda}, \sigma) & \text{if } \frac{2t+\lambda-1}{2\lambda} \leq 1, \\ \texttt{True} & \text{if } \frac{2t+\lambda-1}{2\lambda} > 1. \end{cases}$$

One may observe that $\triangledown$ and $\blacktriangledown$ can actually be defined within the LTL$^{\text{disc}}[\mathcal{D}]$ setting. Indeed, the operator $\triangledown_\lambda$ is similar to a one-time application of $\mathsf{U}_{\exp_\lambda^{+1}}$; thus, $\triangledown_\lambda \varphi$ is equivalent to $\texttt{False}\mathsf{U}_{\exp_\lambda^{+1}}\psi$. We can then use the equivalence $\blacktriangledown_\lambda \varphi \equiv \neg\triangledown_\lambda\neg\varphi$ to express $\blacktriangledown$. As for the $\overline{\blacktriangledown}_\lambda$ operator, it can be similarly expressed using the O operators defined in Section 5.3, as $\overline{\blacktriangledown}_\lambda\varphi$ is equivalent to $\texttt{False}\mathsf{O}_{\exp_\lambda^{+1},\frac{1}{2}}\varphi$.

Finally, note that by combining $\blacktriangledown$ and X, one can define the discounted-next operator mentioned in Remark 4.1.

7. SUMMARY AND DISCUSSION

7.1. Summary

An ability to specify and to reason about quality would take formal methods a significant step forward. Beyond much more informative verification and synthesis procedures, designers would be willing to give up manual design only when automatically synthesized methods would return systems of comparable quality. Quality has many aspects, some of which are propositional, such as prioritizing one satisfaction scheme on top of another, and some are temporal, for example, having higher quality for implementations with shorter delays. In this work, we provided solutions for specifying and reasoning about both propositional and temporal quality by augmenting the commonly used linear temporal logic (LTL).

On the propositional-quality front, our scheme, denoted LTL[$\mathcal{F}$], is based on augmenting LTL with an arbitrary set of functions. In effect, this enables the designer to quantitatively combine subformulas in intricate and nested manners. We showed that, on the one hand, LTL[$\mathcal{F}$] formulas can have exponentially many satisfaction values, which allows one to succinctly represent and rank many ways of satisfying a specification, and on the other hand, model checking LTL[$\mathcal{F}$] can be done in PSPACE, as the complexity for standard LTL.

On the temporal-quality front, our scheme, denoted LTL$^{\text{disc}}[\mathcal{D}]$, is based on augmenting the *until* operator of LTL with an arbitrary set of discounting functions. This enables one to specify how delays in satisfying a requirement influence the level of satisfaction. Such a satisfaction scheme, which is based on elapsed times, introduces a big challenge, as it implies infinitely many satisfaction values. Nonetheless, we showed the decidability of the model-checking problem and, for the natural exponential-decaying satisfactions, the complexity remains as the one for standard LTL, suggesting the interesting potential of the new scheme.

As for combining propositional- and temporal-quality operators, we showed that the problem is, in general, undecidable (even when only simple propositional functions such as average are allowed), while certain combinations, such as adding priorities, preserve the decidability and complexity.

7.2. Future Research

The Expressive Power of LTL$^{disc}[\mathcal{D}]$ *and* PLTL$^{disc}[\mathcal{D}]$. In Section 5.2, we add discounting past operators to LTL$^{\text{disc}}[\mathcal{D}]$, and show that model-checking can be solved with the same complexity as model-checking LTL$^{\text{disc}}[\mathcal{D}]$. In the Boolean setting, it is known that past operators do not add expressive power to LTL (but do allow exponentially more succinct formulas) [Lichtenstein et al. 1985]. The proof that LTL and PLTL have the same expressive power is notoriously complicated, and does not offer a direct translation.

An interesting direction for future research is to compare the expressive power of LTL$^{\text{disc}}[\mathcal{D}]$ and PLTL$^{\text{disc}}[\mathcal{D}]$. We conjecture that PLTL$^{\text{disc}}[\mathcal{D}]$ is strictly more expressive

than $\text{LTL}^{\text{disc}}[\mathcal{D}]$, specifically, that the formula $\mathsf{F}_\eta \neg(\texttt{True}\mathsf{S}_\eta p)$ over the atomic proposition $AP = \{p\}$ does not have an equivalent $\text{LTL}^{\text{disc}}[\mathcal{D}]$ formula.

Fragments of $\text{LTL}[\mathcal{F}]$ *and* $\text{LTL}^{disc}[\mathcal{D}]$. In the Boolean setting, model checking and synthesis suffer from very high complexity. This renders model checking expensive, and synthesis impractical. To deal with this high complexity, researchers suggest looking at fragments of LTL. One particular fragment that is very expressive is GR(1) [Piterman et al. 2006]. For practical applications, it would be useful to look at this fragment in the context of $\text{LTL}[\mathcal{F}]$ and $\text{LTL}^{\text{disc}}[\mathcal{D}]$.

For $\text{LTL}[\mathcal{F}]$, handling propositional operators is somewhat orthogonal to the inherent complexity of model checking. Thus, it would not be hard to reuse techniques that work with fragments of LTL to $\text{LTL}[\mathcal{F}]$. In particular, the $\text{LTL}[\mathcal{F}]$ analogue of the GR(1) fragment allows ranking of the Streett conditions. That is, instead of considering a formula of the form $(\mathsf{GF}\varphi_1 \wedge \ldots \wedge \mathsf{GF}\varphi_n) \rightarrow (\mathsf{GF}\psi_1 \wedge \ldots \wedge \mathsf{GF}\psi_n)$, we consider the formula $(\mathsf{GF}\varphi_1 \wedge \ldots \wedge \mathsf{GF}\varphi_n) \rightarrow f(\mathsf{GF}\psi_1, \ldots, \mathsf{GF}\psi_n)$, where f is a function that ranks different subsets of its inputs. This allows us to prioritize the importance of the goals $\psi_1, \ldots, \psi_n$.

For $\text{LTL}^{\text{disc}}[\mathcal{D}]$, it is less clear that standard GR(1) algorithms can readily be adapted; this is an important direction for future work. The benefit of using $\text{LTL}^{\text{disc}}[\mathcal{D}]$ in GR(1) is that we can strengthen the fairness condition as well as the responses, as follows. Instead of the standard GR(1) formula $(\mathsf{GF}\varphi_1 \wedge \ldots \wedge \mathsf{GF}\varphi_n) \rightarrow (\mathsf{GF}\psi_1 \wedge \ldots \wedge \mathsf{GF}\psi_n)$, we can use the formula $(\mathsf{GF}_\eta\varphi_1 \wedge \ldots \wedge \mathsf{GF}_\eta\varphi_n) \rightarrow (\mathsf{GF}_\eta\psi_1 \wedge \ldots \wedge \mathsf{GF}_\eta\psi_n)$, which requires the fairness condition and the responses to hold not only infinitely often, but also in small intervals, which is often desirable.

Another fragment that is especially relevant for $\text{LTL}^{\text{disc}}[\mathcal{D}]$ is of formulas with bounded nesting. In practical settings, the nesting depth of LTL formulas is typically very low. Assuming a bounded nesting depth in $\text{LTL}^{\text{disc}}[\mathcal{D}]$ will also entail simplifications in our constructions. In particular, observe that the construction in Lemma 4.11 is exponential in the nesting depth of the formula (but also in the description of the discounting factors). Thus, if the nesting depth is bounded, the construction becomes much smaller.

REFERENCES

Eric Allender, Peter Bürgisser, Johan Kjeldgaard-Pedersen, and Peter Bro Miltersen. 2009. On the complexity of numerical analysis. *SIAM Journal on Computing* 38, 5, 1987–2006.

Shaull Almagor, Guy Avni, and Orna Kupferman. 2013a. Automatic generation of quality specifications. In *Proceedings of the 25th International Conference on Computer Aided Verification*. Lecture Notes in Computer Science, Vol. 8044. Springer, Berlin, 479–494.

Shaull Almagor, Udi Boker, and Orna Kupferman. 2011. What's decidable about weighted automata? In *9th International Symposium on Automated Technology for Verification and Analysis*. Lecture Notes in Computer Science, Vol. 6996. Springer, Berlin, 482–491.

Shaull Almagor, Udi Boker, and Orna Kupferman. 2013b. Formalizing and reasoning about quality. In *Proceedings of the 40th International Colloqium on Automata, Languages, and Programming*. Lecture Notes in Computer Science, Vol. 7966. Springer, Berlin, 15–27.

Shaull Almagor, Udi Boker, and Orna Kupferman. 2014. Discounting in LTL. In *Proceedings of the 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Lecture Notes in Computer Science, Vol. 8413. Springer, Berlin 424–439.

Shaull Almagor, Yoram Hirshfeld, and Orna Kupferman. 2010. Promptness in omega-regular automata. In *8th International Symposium on Automated Technology for Verification and Analysis*. Lecture Notes in Computer Science, Vol. 6252. Springer, Berlin, 22–36.

Shaull Almagor and Orna Kupferman. 2011. Max and sum semantics for alternating weighted automata. In *9th International Symposium on Automated Technology for Verification and Analysis*. Lecture Notes in Computer Science, Vol. 6996. Springer, Berlin, 13–27.

Shaull Almagor and Orna Kupferman. 2015. High-quality synthesis against stochastic environments *(Submitted)*.

Sanjeev Arora and Boaz Barak. 2009. *Computational Complexity—A Modern Approach*. Cambridge University Press, New York, NY.

Roderick Bloem, Krishnendu Chatterjee, Thomas A. Henzinger, and Barbara Jobstmann. 2009. Better quality in synthesis through quantitative objectives. In *Proceedings of the 21st International Conference on Computer Aided Verification*. Lecture Notes in Computer Science, Vol. 5643. Springer, Berlin, 140–156.

A. Bohy, V. Bruyère, E. Filiot, and J.-F. Raskin. 2013. Synthesis from LTL specifications with mean-payoff objectives. In *Proceedings of the 19th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Lecture Notes in Computer Science, Vol. 7795. Springer, Berlin, 169–184.

Mikolaj Bojańczyk and Thomas Colcombet. 2006. Bounds in ω-regularity. In *Proceedings of the 21st IEEE Symposium on Logic in Computer Science*. 285–296.

Udi Boker, Krishnendu Chatterjee, Thomas A. Henzinger, and Orna Kupferman. 2014. Temporal specifications with accumulative values. *ACM Transactions on Computational Logic* 15, 4, 27:1–27:25.

Udi Boker, Thomas A. Henzinger, and Jan Otop. 2015. The target discounted-sum problem. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS'15)*, Kyoto, Japan. 750–761.

Udi Boker, Orna Kupferman, and Adin Rosenberg. 2010. Alternation removal in Büchi automata. In *Proceedings of the 37th International Colloquium on Automata, Languages, and Programming*. Lecture Notes in Computer Science, Vol. 6199. Springer, Berlin, 76–87.

Patricia Bouyer, Nicolas Markey, and Raj Mohan Matteplackel. 2014. Averaging in LTL. In *Proceedings of CONCUR 2014 - Concurrency Theory - 25th International Conference (CONCUR'14)*, Rome, Italy. 266–280.

Glenn Bruns and Patrice Godefroid. 2004. Model checking with multi-valued logics. In *Proceedings of the 31st International Colloquium on Automata, Languages, and Programming*. Lecture Notes in Computer Science, Vol. 3142. 281–293.

Pavol Černỳ, Krishnendu Chatterjee, Thomas A. Henzinger, Arjun Radhakrishna, and Rohit Singh. 2011. Quantitative synthesis for concurrent programs. In *Proceedings of the 23rd International Conference on Computer Aided Verification*. 243–259.

Krishnendu Chatterjee, Vojtech Forejt, and Dominik Wojtczak. 2013. Multi-objective discounted reward verification in graphs and MDPs. In *Proceedings of the 19th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'19), Stellenbosch, South Africa*. 228–242.

Edmund M. Clarke, Orna Grumberg, and Doron Peled. 1999. *Model Checking*. MIT Press, Cambridge, MA.

Mads Dam. 1994. CTL* and ECTL* as fragments of the modal μ-calculus. *Theoretical Computer Science* 126, 77–96.

Luca De Alfaro, Marco Faella, Thomas A. Henzinger, Rupak Majumdar, and Mariëlle Stoelinga. 2005. Model checking discounted temporal properties. *Theoretical Computer Science* 345, 1, 139–170.

Luca De Alfaro, Marco Faella, and Mariëlle Stoelinga. 2004. Linear and branching metrics for quantitative transition systems. In *Proceedings of the 31st International Colloquium on Automata, Languages, and Programming*. Lecture Notes in Computer Science, Vol. 3142. 97–109.

Luca De Alfaro, Thomas A. Henzinger, and Rupak Majumdar. 2003. Discounting the future in systems theory. In *Proceedings of the 30th International Colloquium on Automata, Languages, and Programming*. Lecture Notes in Computer Science, Vol. 2719. 1022–1037.

Josée Desharnais, Vineet Gupta, Radha Jagadeesan, and Prakash Panangaden. 2004. Metrics for labelled Markov processes. *Theoretical Computer Science* 318, 3, 323–354.

Alexandre Donzé and Oded Maler. 2010. Robust satisfaction of temporal logic over real-valued signals. In *Proceedings of the 8th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'10)*. 92–106.

Alexandre Donzé, Oded Maler, Ezio Bartocci, Dejan Nickovic, Radu Grosu, and Scott Smolka. 2012. On temporal logic and signal processing. In *Proceedings of the 10th International Conference on Automated Technology for Verification and Analysis (ATVA'12)*. Springer, Berlin, 92–106.

M. Droste, W. Kuich, and H. Vogler (eds.). 2009. *Handbook of Weighted Automata*. Springer.

Manfred Droste, Werner Kuich, and George Rahonis. 2008. Multi-valued MSO logics OverWords and trees. *Fundammenta Informaticae* 84, 3–4, 305–327.

Manfred Droste and George Rahonis. 2009. Weighted automata and weighted logics with discounting. *Theoretical Computer Science* 410, 37, 3481–3494.

Manfred Droste and Heiko Vogler. 2012. Weighted automata and multi-valued logics over arbitrary bounded lattices. *Theoretical Computer Science* 418, 14–36.

E. Allen Emerson and Joseph Y. Halpern. 1986. Sometimes and not never revisited: On branching versus linear time. *Journal of the ACM* 33, 1, 151–178.

E. Allen Emerson and Chin-Laung Lei. 1985. Modalities for model checking: Branching time logic strikes back. In *Proc. 12th ACM Symp. on Principles of Programming Languages*. 84–96.

E. Allen Emerson and Chin-Laung Lei. 1986. Efficient model checking in fragments of the propositional μ-calculus. In *Proceedings of the 1st IEEE Symposium on Logic in Computer Science*. 267–278.

Marco Faella, Axel Legay, and Mariëlle Stoelinga. 2008. Model checking quantitative linear time logic. *Electronic Notes in Theoretical Computer Science* 220, 3, 61–77.

Emmanuel Filiot, Raffaella Gentilini, and Jean-François Raskin. 2014. Finite-valued weighted automata. In *34th International Conference on Foundation of Software Technology and Theoretical Computer Science, (FSTTCS'14)*, New Delhi, India. 133–145.

Martin Fränzle, Michael R. Hansen, and Heinrich Ody. 2015. Discounted duration calculus. In *Proceedings of the 27th Nordic Workshop on Programming Theory*. RUTR-SCS16001, 75–77.

Paul Gastin and Denis Oddoux. 2001. Fast LTL to Büchi automata translation. In *Proceedings of the 13th International Conference on Computer Aided Verification.* Lecture Notes in Computer Science, Vol. 2102. Springer, Berlin, 53–65.

Y. Gurevich and L. Harrington. 1982. Trees, automata, and games. In *Proceedings of the 14th ACM Symposium on Theory of Computing*. ACM Press, New York, NY, 60–65.

Hans Hansson and Bengt Jonsson. 1994. A logic for reasoning about time and reliability. *Formal Aspects of Computing* 6, 102–111.

Gerard J. Holzmann. 2004. *The Spin Model Checker: Primer and Reference Manual*. Addison-Wesley, New York, NY.

IEEE. 1993. IEEE Standard Multivalue Logic System for VHDL Model Interoperability (Std_logic_1164).

Stephen H. Kan. 2002. *Metrics and Models in Software Quality Engineering* (2nd ed.). Addison-Wesley Longman Publishing Co., Inc., New York, NY.

Daniel Kirsten and Sylvain Lombardy. 2009. Deciding unambiguity and sequentiality of polynomially ambiguous min-plus automata. In *Proceedings of the 26th International Symposium on Theoretical Aspects of Computer Science (STACS'09)*. Freiburg, Germany, 589–600.

Daniel Krob. 1994. The equality problem for rational series with multiplicities in the tropical semiring is undecidable. *International Journal of Algebra and Computation* 4, 3, 405–425.

Orna Kupferman. 2006. Sanity checks in formal verification. In *Proceedings of the 17th International Conference on Concurrency Theory.* Lecture Notes in Computer Science, Vol. 4137. Springer, Berlin, 37–51.

Orna Kupferman and Yoad Lustig. 2007. Lattice automata. In *Proceedings of the 8th International Conference on Verification, Model Checking, and Abstract Interpretation.* Lecture Notes in Computer Science, Vol. 4349. Springer, Berlin, 199–213.

Orna Kupferman, Nir Piterman, and Moshe Y. Vardi. 2006. Safraless compositional synthesis. In *Proceedings of the 18th International Conference on Computer Aided Verification.* Lecture Notes in Computer Science, Vol. 4144. Springer, Berlin, 31–44.

Orna Kupferman, Nir Piterman, and Moshe Y. Vardi. 2009. From liveness to promptness. *Formal Methods in System Design* 34, 2, 83–103.

Orna Kupferman and Moshe Y. Vardi. 1997. Synthesis with incomplete information. In *2nd International Conference on Temporal Logic*. 91–106.

Orna Kupferman and Moshe Y. Vardi. 2005. Safraless decision procedures. In *Proceedings of the 46th IEEE Symposium on Foundations of Computer Science*. 531–540.

Orna Kupferman, Moshe Y. Vardi, and Pierre Wolper. 2000. An automata-theoretic approach to branching-time model checking. *Journal of the ACM* 47, 2, 312–360.

Robert P. Kurshan. 1998. *FormalCheck User's Manual*. Cadence Design Systems, Inc. Columbia, MD.

Marta Z. Kwiatkowska. 2007. Quantitative verification: Models techniques and tools. In *ESEC/SIGSOFT FSE*. 449–458.

François Laroussinie and Ph Schnoebelen. 1994. A hierarchy of temporal logics with past. In *Proceedings of the 11th Symposium on Theoretical Aspects of Computer Science*. 47–58.

Orna Lichtenstein, Amir Pnueli, and Lenore Zuck. 1985. The glory of the past. In *Logics of Programs.* Lecture Notes in Computer Science, Vol. 193. Springer, Berlin, 196–218.

Eleni Mandrali. 2012. Weighted LTL with discounting. In *CIAA.* Lecture Notes in Computer Science, Vol. 7381. Springer, Berlin, 353–360.

Z. Manna and A. Pnueli. 1995. *The Temporal Logic of Reactive and Concurrent Systems: Safety*. Springer.

Marvin L. Minsky. 1967. *Computation: Finite and Infinite Machines* (1 ed.). Prentice Hall, Upper Saddle River, NJ.

Satoru Miyano and Takeshi Hayashi. 1984. Alternating finite automata on ω-words. *Theoretical Computer Science* 32, 321–330.

Mehryar Mohri. 1997. Finite-state transducers in language and speech processing. *Computational Linguistics* 23, 2, 269–311.

Seong-ick Moon, Kwang Hyung Lee, and Doheon Lee. 2004. Fuzzy branching temporal logic. *IEEE Transactions on Systems, Man, and Cybernetics, Part B* 34, 2, 1045–1055.

Shota Nakagawa and Ichiro Hasuo. 2015. Near-optimal scheduler synthesis for LTL with future discounting. In *10th International Symposium on Trustworthy Global Computing*.

N. Piterman, A. Pnueli, and Y. Saar. 2006. Synthesis of reactive(1) designs. In *Proceedings of the 7th International Conference on Verification, Model Checking, and Abstract Interpretation.* Lecture Notes in Computer Science, Vol. 3855. Springer, Berlin, 364–380.

Amir Pnueli and Roni Rosner. 1989. On the synthesis of a reactive module. In *Proceedings of the 16th ACM Symposium on Principles of Programming Languages*. 179–190.

Claude E. Shannon. 1949. The synthesis of two terminal switching circuits. 28, 1, 59–98.

Lloyd S. Shapley. 1953. Stochastic games. In *Proceedings of the National Academy of Science*, Vol. 39. National Academy of Sciences, 1095.

Diomidis Spinellis. 2003. *Code Reading: The Open Source Perspective*. Addison-Wesley, New York, NY.

Wolfgang Thomas. 1990. Automata on infinite objects. *Handbook of Theoretical Computer Science* 2, 133–191.

Moshe Y. Vardi. 1996. An automata-theoretic approach to linear temporal logic. In *Logics for Concurrency: Structure versus Automata.* Lecture Notes in Computer Science, F. Moller and G. Birtwistle (Eds.), Vol. 1043. Springer, Berliln, 238–266.

Moshe Y. Vardi and Pierre Wolper. 1986. An automata-theoretic approach to automatic program verification. In *Proceedings of the 1st IEEE Symposium on Logic in Computer Science*. 332–344.

Moshe Y. Vardi and Pierre Wolper. 1994. Reasoning about infinite computations. *Information and Computation* 115, 1, 1–37.

Received November 2014; revised November 2015; accepted January 2016