Formalizing and Reasoning about Quality

Thesis for the degree of
"Doctor of Philosophy"

by

Shaull Almagor

Submitted to the Senate of the Hebrew University of Jerusalem

May, 2016

This work was carried out under the supervision of Orna Kupferman.

Acknowledgements

Prof. Orna Kupferman, my advisor, writes in her PhD acknowledgments: “I believe that those whom I want to thank know exactly how deep my thanks are”. I also believe that, but if there is one lesson I learned during my studies, is that even when we believe something to be true, it is still important to write down all the details.

First and foremost, I want to thank Orna. The work with her was challenging, interesting, educating, and always fun. I felt free to research in my own pace, yet her guidance, ranging from showing the overall picture, through suggesting technical ideas, and up to pointing out a rogue comma in an obscure appendix, was a driving force whenever I needed it. The depth of my learning from Orna cannot be overstated. I often re-read email correspondences with her from months, or even years ago, and learn something new.

This also seems the place to acknowledge the huge impact Orna had on the course of my life. During my undergrad studies, I wanted to focus on computer graphics. However, I first had to take Orna’s computability course. There, I was enchanted by Orna’s teaching, and by theoretical computer science. It was this that led me to take Orna’s course on automata, and eventually to do my MSc under her supervision (graphics was long forgotten). The decision to continue to a PhD under Orna’s supervision was obvious, on my part. During the first year of my PhD, Orna told me about a mathematical-education program, led by prof. Raz Kupferman, in which graduate students teach mathematics to elementary-school teachers. Orna allowed me to join this program, where my already-existing passion for teaching grew. I thank both her and Raz for this opportunity. Later on, I was offered to teach high-school mathematics. Orna approved of me taking this position, even though it was quite time-consuming. Apart from enjoying and learning a lot from teaching, it is there that I met my wife, and for that I am most grateful.

Several works in my PhD were done in collaboration with Udi Boker, who was Orna’s postdoc at the time. Udi’s quick thinking made working with him exciting and extremely productive, while his cheerfulness made this work enjoyable. I learned much from his meticulous style of work (i.e. I found bugs in my proofs).

I want to thank Denis Kuperberg, with whom we worked on sensing. His bold dives into technical messes saved us several times.

I also want to thank Prof. Moshe Vardi and his group at Rice University. Every meeting with Moshe provided a new point of view at things I already thought I knew. Collaborating with his group introduced me to the field of robotic planning and its relation to formal methods.

My great thanks to Guy Avni, with whom I’ve shared an office for the past seven years. We’ve been together to numerous conferences, shared hotel rooms, beds, meals, drinks, near-death experiences, and even published two papers together. Guy is always in good humor, and I hope our paths will cross again in the future.

My teaching colleague these past five years is Jonathan Mosheiff. My dream-job since the second year of undergraduate studies was to teach computability. I got this job, and I could not have asked for a better colleague in it. I hope our end-of-year songs will remain a legacy.

Much of my research is done at home, and therefore I consider its inhabitants a crucial part of my academic career. My thanks to Tali, my wife, who was also my colleague as a teacher: for her absolute support, for her ability to share my frustration, and for uprooting her life to follow me to a postdoc. My thanks also to Gamma, my dog. Most of the solutions for the problems in this thesis were thought of during our walks.

Last but not least, my thanks to my mother, for bringing me up with a passion to learn and to teach, for always having interesting conversations with me, and for listening to countless practice talks and lectures.

People often say that looking back, the years in the university were the best time of their life. I think I can say that without looking back.

Abstract

Traditional formal methods tackle two main problems, namely *model checking* and *synthesis*. In model checking, we are given a system and a specification, and we wish to check whether the system satisfies the specification. In synthesis, we are only given a specification, and our goal is to automatically construct a correct system – one that satisfies the specification.

Correctness is Boolean: a system can either satisfy its specification or not satisfy it. The richness of today's systems, however, justifies formalisms that are *quantitative*.

In this thesis, we lift the model-checking and the synthesis problems to the quantitative settings, and study them, as well as other problems that arise in the quantitative setting. Thus, we reason not only about *whether* a system satisfies its specification, but also about the quality of satisfaction. That is, *how well* the system satisfies the specification.

We start by introducing specification formalisms that allow the designer to specify the quality of a system by means of *quality operators*.

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.

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.

Our next contribution considers an extension of LTL to a multi-valued domain, namely a lattice. In *latticed linear temporal logic* (LLTL), conjunctions and disjunctions correspond to the meet and join operators of the lattice $\mathcal{L}$, respectively. An LLTL formula over the atomic propositions AP is evaluated over lattice-valued computations, i.e. computations in $(AP^{\mathcal{L}})^\omega$, and the satisfaction value of a formula in a computation is thus also an element of the lattice. The lattice setting arises in practice, for example in specifications involving priorities or in systems with inconsistent viewpoints.

We solve the LLTL synthesis problem, where the goal is to synthesize a transducer that realizes the given specification in a desired satisfaction value.

For the classical synthesis problem, researchers have studied a setting with incomplete information, where the truth values of some of the input signals are hidden and the transducer should nevertheless realize a specification ψ. For the multi-valued setting, we introduce and study a new type of incomplete information, where the truth values of some

of the input signals may be *noisy*, and the transducer should still realize ψ in the desired satisfaction value. We study the problem of noisy LLTL synthesis, as well as the theoretical aspects of the setting, like the amount of noise a transducer may tolerate, or the effect of perturbing input signals on the satisfaction value of a specification.

The last two contributions in the thesis consider a different aspect of quality – one where the measure of quality stems directly from the system. We introduce and study a new complexity measure for regular languages, namely the *sensing cost* of a language. Intuitively, the sensing cost of a language measures the detail with which a random input word needs to be read in order to decide membership in the language.

We show that for finite words, minimizing the state space of a DFA can only reduce its sensing cost. In particular, since DFAs can be minimized in polynomial time, we can construct in polynomial time a minimally-sensing DFA, and can compute in polynomial time the sensing cost of languages given by DFAs.

We then study the sensing cost of ω-regular languages, given by means of deterministic parity automata (DPAs). We show that unlike the case of finite words, the minimal sensing may be attained only by an infinite sequence of automata. We show that the optimal limit cost of such sequences can be characterized by the language's right-congruence relation, which enables us to find the sensing cost of ω-regular languages in polynomial time.

Next, we restrict attention to monitoring. There, we are only concerned about the sensing cost of words that are in the language. Accordingly, we develop new and different frameworks for reasoning about the sample space of words in the language. We show that for monitoring, minimal sensing is attained by a monitor based on the minimal deterministic automaton for the language.

Finally, we study sensing in the synthesis setting. We show that there, the situation is more challenging: minimizing the sensing cost of a specification may require exponentially bigger transducers, and the problem of synthesizing a minimally-sensing transducer is EXPTIME-complete even for safety specifications given by deterministic automata.

Letter Indicating the Contribution to Each Chapter

All four works in this thesis were carried out under the supervision of Orna Kupferman, with me as principal researcher.

The first work is a joint work with Udi Boker, who was a post-doctoral student of Orna Kupferman. Udi was involved in all stages of the work, contributing by fruitful discussions, as well as directly contributing to some of the results (in particular, to Theorems 2.2, 6.3, and 6.4).

In the second work I am the sole researcher, under the supervision of Orna Kupferman.

The third and fourth works are joint with Denis Kuperberg, who was a post-doctoral student of Orna Kupferman. Denis provided valuable expertise in parity automata and analytic combinatorics. In the third work, Denis contributed mainly to the proofs in Section 4.1, and in the fourth work, to the proof of Theorem 3.2.

Contents

1 Introduction

Designing and implementing complex computerized systems (e.g., software or hardware) is an error-prone task. The central role that such systems play in our lives makes it crucial that systems are bug-free. In the past few decades, we have witnessed a growing demand for formal, mathematically-proven methods to ensure the correctness of systems, and to automatically design correct systems. These demands were met in academia by existing theoretical paradigms, such as the rich theory of automata, and led to the development of many successful theoretical frameworks, as well as industrial-strength tools [29, 31, 36].

The success of formal methods, and the increasingly complex structure of today's systems, motivated researchers to look for extensions and improvements of the current methods. One new and exciting extension is the quantitative setting, in which we reason not only about the (Boolean) correctness of systems, but about quantitative properties of systems. For example, instead of asking *whether* a system satisfies a specification, we may ask *with what quality* it satisfies the specification.

This quantitative notion of quality is inherently subjective. Indeed, the level of quality of a system depends on the aspects we measure (e.g., speed v.s. power consumption). Moreover, quantitativeness may arise from different sources: from the system, from the specification, or from a combination of the two. The focus of this thesis is to formalize and to reason about quality, in different settings and with various sources.

We demonstrate the focus of this thesis with an example.

Example 1.1 Consider a system that receives requests and responds with grants. A sensible Boolean specification for the correctness of such a system is that every request is eventually followed by a grant. We now consider three systems that satisfy this specification. System A always waits for a request, and grants a request in the next step. System B always outputs grants, regardless of requests, and System C, once a request is given, idles for 100 steps, and then outputs a grant and waits for another request.

Clearly all three systems satisfy the Boolean specification. However, we can consider several quantitative parameters that differentiate between them. We start by considering the response time of the systems. System B satisfies every request immediately, whereas System A responds after one time unit, and System C may wait as long as a 100 steps. We can thus rank the systems according to their response time. Next, assume that requests are received uniformly at random. That is, in every step a request is received with probability $\frac{1}{2}$. Additionally, assume that reading a request involves activating some sensor, which incurs a cost. We can now measure the average expected *sensing cost* of the systems. System B never senses anything, so its cost is 0. System A senses at every step, so its cost is 1. As for System C, once a request is given, sensing is paused for 100 steps, so the sensing cost is a number $0 < p < 1$. Finally, we observe that while System B indeed satisfies the Boolean specification, it does so *vacuously*, in the sense that it outputs grants even if no requests are given. In an adequate quantitative specification formalism, we can explicitly penalize such behavior by giving lower quality to systems that output grants when no requests are given.

The measure of response time is a quantitative specification: different ways of satisfying this specification induce different values. Similarly, the penalty for vacuous satisfaction stems from the specification. The sensing cost, in contrast, stems from the system, and is independent of the specification. □

Traditional formal methods tackle two main problems, namely *model checking* and *synthesis*. In model checking, we are given a system and a specification, and we wish to check whether the system satisfies the specification [28]. In *synthesis*, we are only given a specification, and our goal is to automatically construct a system that satisfies the specification [46]. In this thesis, we lift these problems to quantitative settings, and study them, as well as other problems that arise in the quantitative setting.

We now turn to present our contribution.

Temporal Logic with Quality Operators.

In the Boolean setting, one of the most popular specification formalisms is *Linear Temporal Logic* (LTL, for short) [28], which is derived from propositional logic by adding temporal operators, like G ("always") and F("eventually"), which describe a temporal ordering of events. For example, the LTL formula $\mathsf{G}(req \to \mathsf{F}grant)$ specifies that after every request there is eventually a grant.

In this context, a finite-state system is modeled by a labeled state-transition graph. Then, the model-checking problem amounts to checking whether every computation of the system satisfies the specification, and the synthesis problem amounts to generating a system from an LTL formula.

The logic LTL is Boolean, in the sense that an LTL formula maps a computation of a system to either `True` or `False`. In the following works, we study augmentations of LTL that allow the specification of quantitative properties. The systems we consider remain Boolean. Thus, the source of quantitativeness in these works is the specification alone.

Propositional Quality

In this work, we introduce a family of quantitative temporal logics $\mathrm{LTL}[\mathcal{F}]$, parametrized by a set $\mathcal{F}$ (which will be explained shortly), in which a formula ψ maps a computation π to a value in $[0,1]$ indicating the quality of the satisfaction of ψ in π. The parameter $\mathcal{F}$ is a set $\mathcal{F} \subseteq \{f : [0,1]^k \to [0,1] \mid k \in \mathbb{N}\}$ of arbitrary functions that are used to describe quantitative relations between subformulas. Observe that the functions may have different arities, which allows the specification to relate multiple subformulas in complex ways.

For example, the formula $\psi_1 \oplus_{\frac{2}{3}} \psi_2$, with the *weighted average* operator $\oplus_{\frac{2}{3}}$, specifies that the quality of a computation π is the value $\frac{2}{3}x + \frac{1}{3}y$, where x and y are the satisfaction values of the formulas ψ_1 and ψ_2 in π, respectively. In this example, the set $\mathcal{F}$ contains the weighted average operator. This can be used, for example, when specifying the quality of a system that consists of two servers, where Server 1 performs twice as much of the work as Serve 2, and ψ_1 and ψ_2 specify the quality of servers 1 and 2, respectively.

Other useful operators include $\triangledown_\lambda \varphi$ for $\lambda \in (0, 1)$, $\neg\varphi$ and $\varphi \wedge \psi$, which assign satisfaction values λx, $1-x$ and $\min\{x, y\}$, respectively, where x and y are the satisfaction values of φ and ψ. We demonstrate their use in the following example. Consider the LTL formula $\mathsf{G}(req \rightarrow \mathsf{X}grant \wedge \mathsf{XX}grant)$, which states that every request should be followed by two consecutive grants. One may consider also allowing a grant to be given for only a single step, at the cost of reducing the satisfaction value. We achieve this with the LTL$[\mathcal{F}]$ formula $\mathsf{G}(req \rightarrow (\mathsf{X}grant) \oplus_{\frac{1}{2}} (\mathsf{XX}grant))$. The satisfaction value of the formula is 1 if every request is eventually granted, and 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, we want to penalize systems that do not generate requests, and thus satisfy the specification vacuously. We achieve this with the LTL$[\mathcal{F}]$ formula $\mathsf{G}(req \rightarrow (\mathsf{X}grant) \oplus_{\frac{1}{2}} (\mathsf{XX}grant)) \wedge \neg(\triangledown_{\frac{3}{4}} \mathsf{G}\neg req)$. Here, 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.

We dub this approach *propositional quality*, as the functions in $\mathcal{F}$ play a propositional role. That is, they do not induce new temporal operators. We study this logic, as well as the model-checking and the synthesis problem for it. Observe that since the satisfaction value of an LTL$[\mathcal{F}]$ formula is a value in $[0, 1]$, the model-checking problem becomes a search problem, rather than a decision problem (as it is in the Boolean setting). Specifically, given a system $\mathcal{K}$ and an LTL$[\mathcal{F}]$ formula φ, the model-checking problem is to calculate $[\![\mathcal{K}, \varphi]\!]$ – the minimal satisfaction value of a computation of $\mathcal{K}$ in φ. Similarly, the synthesis problem is to output, given an LTL$[\mathcal{F}]$ formula φ, a transducer $\mathcal{K}$ that is receptive to all input sequences, and maximizes $[\![\mathcal{K}, \varphi]\!]$.

We show how to solve the model-checking and synthesis problems for this logic, in the same complexity classes as their analogues in the Boolean setting. Our solution uses a translation to Boolean automata, which are used in many industrial-strength tools. This implies that our algorithms can be easily incorporated into existing tools. In particular, solving LTL$[\mathcal{F}]$ synthesis implies that we can generate high-quality systems.

Specifically, our construction is based on the Vardi-Wolper construction [50]. There, an LTL formula is translated to a *nondeterministic Büchi automaton* (NBW, for short) such that each state of the NBW is associated with a set of sub-formulas, and the automaton accepts a computation from a state q iff the computation satisfies exactly all the formulas associated with q.

In our construction, each state of the NBW assigns a satisfaction *value* to every subformula. The states are chosen to be consistent, which assures that the satisfaction values agree with the functions in $\mathcal{F}$. Similar adjustments are made to the transitions and the acceptance condition. A-priori, this construction is infinite, since we need to assign to every subformula every value in $[0,1]$. However, we prove that every $\text{LTL}[\mathcal{F}]$ formula φ has a finite set of possible satisfaction values. This enables us to construct only finitely many states in the translation to an NBW.

These results have been described in [4, 6], and appear in Chapter 2 of this thesis.

Temporal Quality

An orthogonal approach to propositional quality is to take into consideration the temporal aspects of satisfaction. One way to do so is *future discounting*, in which events in the far future have a smaller effect on the satisfaction value of formulas. For example, recall the formula $\mathsf{G}(req \to \mathsf{F}\ grant)$. This formula can be satisfied by a system that takes long delays in issuing grants (e.g., System C in Example 1.1), or by a system that has unbounded delays, which is an undesired behavior.

To address this issue, we suggest a family of logics $\text{LTL}^{\text{disc}}[\mathcal{D}]$, parametrized by a set $\mathcal{D}$ of *discounting functions*. The logic $\text{LTL}^{\text{disc}}[\mathcal{D}]$ augments LTL with future discounting operators such as F_η, where $\eta \in \mathcal{D}$ is a discounting function – a strictly decreasing function $\eta : \mathbb{N} \to [0,1]$ such that $\lim_{n\to\infty} \eta(n) = 0$. For example, the formula $\mathsf{G}(req \to \mathsf{F}_\eta grant)$, where $\eta(i) = 0.9^i$, is an $\text{LTL}^{\text{disc}}[\mathcal{D}]$ formula. The satisfaction value of this formula in a path π is 0.9^k, where k is the maximal delay between a request and a corresponding grant. Thus, the satisfaction value is decreased when the waiting time for grants is increased, which captures the intuition that eventualities should be satisfied quickly.

Unlike $\text{LTL}[\mathcal{F}]$ formulas, an $\text{LTL}^{\text{disc}}[\mathcal{D}]$ formula may have infinitely many possible satisfaction values. This poses a major obstacle in constructions such as the one used for propositional quality. We study the threshold model-checking problem for $\text{LTL}^{\text{disc}}[\mathcal{D}]$ – given a system $\mathcal{K}$, an $\text{LTL}^{\text{disc}}[\mathcal{D}]$ formula φ, and a threshold $v \in [0,1]$, decide whether every computation of $\mathcal{K}$ satisfies φ with value greater than v.

Our solution uses again the automata-theoretic approach, and consists of the following steps. Using a construction based on that of [51], we start by translating φ and v to a Boolean alternating automaton $\mathcal{A}_{\varphi,v}$ such that the language of $\mathcal{A}_{\varphi,v}$ is non-empty iff there exists a computation π that satisfies φ with value greater than v. Since φ might have infinitely many satisfaction values, naively we would get 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. We note that in contrast to $\text{LTL}[\mathcal{F}]$, this automaton does not capture exactly the set of computations that satisfy φ with value greater than v. Indeed, the latter set may not be ω-regular. However, for a lasso-shaped computation π, the automaton accepts π iff π satisfies φ with value greater than v. We prove that focusing on lasso-shaped computations is sufficient.

We then manipulate this automaton along with the system $\mathcal{K}$ in order to solve the threshold model-checking problem.

In general, the size of the automaton we construct depends on the discounting (the slower the discounting is, the more states we need). Thus, we cannot get a general complexity bound. However, we show that for the common exponential discounting functions, i.e. discounting of the form $\eta(i) = \lambda^i$ for some $\lambda \in (0, 1)$, we maintain the complexity of the Boolean analogue, namely PSPACE. The proof, however, requires intricate representation of the automaton using arithmetic circuits.

These results have been described in [5, 6], and appear in Chapter 2 of this thesis.

Combining $\mathrm{LTL}[\mathcal{F}]$ and $\mathrm{LTL}^{\mathbf{disc}}[\mathcal{D}]$

The positive results in solving the model-checking problem in $\mathrm{LTL}[\mathcal{F}]$ and $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$ motivates us to consider a combination of the two logics. We consider such an extension, and show that even under seemingly very weak conditions, the model-checking problem (and related problems) becomes undecidable. Specifically, we show undecidability for $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$ with a single discounting function η, augmented with the average operator $\oplus_{\frac{1}{2}}$.

Let us explain the idea behind our undecidability proof. Consider the computation $\pi = a^i b^j \#^\omega$. We use temporal discounting to construct formulas $CountA$ and $CountB$ whose satisfaction values in π are $\eta(i)$ and $\eta(j)$, respectively. Let $CompareAB := (CountA \oplus_{\frac{1}{2}} \neg CountB) \wedge (\neg CountA \oplus_{\frac{1}{2}} CountB)$. Then the satisfaction value of $CompareAB$ in π is $\min\left\{\frac{\eta(i)+1-\eta(j)}{2}, \frac{\eta(j)+1-\eta(i)}{2}\right\} = 1 - \frac{|\eta(i)-\eta(j)|}{2}$. Observe that the latter is 1 iff $i = j$, and is less than 1 otherwise. This is because η is strictly decreasing, and in particular an injection. Thus, the formula $CompareAB$ enables us to compare two counters. This ability can be used to reduce the halting problem for two-counter machines, which is known to be undecidable [44], to the model-checking problem for $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$ augmented with $\oplus_{\frac{1}{2}}$. On the positive side, we show that augmenting $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$ with the ∇_λ operators maintains the decidability of the model-checking problem.

These results have also been described in [5, 6], and appear in Chapter 2 of this thesis.

Automatic Generation of Quality Specifications

Already in the Boolean setting, both model checking and synthesis rely on the specification to accurately reflect the designer's intention. One of the criticisms against formal method is that the latter challenge, of coming up with correct specifications, is not much easier than model checking or synthesis. Thus, formal methods merely shift the difficulty of developing correct implementations to that of developing correct specifications [33].

One approach to address this problem is that of automatic generation of specifications. This includes ideas from *learning*, where specifications given by means of automata are learned from a sample of behaviors tagged as good or bad [12]. We introduce a quantitative analogue, and study the problem of automatically generating the quality layer in a particular instance of $\mathrm{LTL}[\mathcal{F}]$, namely $\mathrm{LTL}^{\triangledown}$, where $\mathcal{F} = \{\triangledown_\lambda : \lambda \in (0,1)\}$.

The design of an $\mathrm{LTL}^{\triangledown}$ formula typically starts with an LTL formula on top of which the designer adds the parameterized $\triangledown$ operators. The underlying assumption behind our approach is that in practice it is easier for a designer to provide desired satisfaction values in representative computations than to come up with quality constants that capture his intuition of high and low quality.

Thus, an $\mathrm{LTL}^{\triangledown}$ *query* is an $\mathrm{LTL}^{\triangledown}$ formula φ in which some of the quality constants are replaced by variables. We are then given a set of *path constraints* – pairs of the form $\langle \pi, I \rangle$, where π is a computation and $I \subseteq [0,1]$ is a closed interval. Then, the $\mathrm{LTL}^{\triangledown}$ query problem is to find an assignment to the variables in φ so that all the constraints are satisfied (or return that no such assignment exists).

We start by showing that in general, the $\mathrm{LTL}^{\triangledown}$ query problem is NP-hard, and proceed to study a fragment of the problem, where the $\mathrm{LTL}^{\triangledown}$ queries are simple and the set of constraints includes a single computation. We show that in this case, the problem can be solved in polynomial time.

Finally, we use the case of a single constraint in a heuristic for the general case. We implemented our algorithms and examined the quality constants generated for various specifications.

These results have been described in [1], and do not appear in the thesis due to lack of space.

Latticed-LTL Synthesis in the Presence of Noisy Inputs

The modern approach to synthesis was initiated by Pnueli and Rosner, who introduced LTL synthesis [46]: We are given an LTL formula ψ over sets I and O of input and output signals, and we synthesize a finite-state system that *realizes* ψ. At each moment in time, the system reads a truth assignment, generated by the environment, to the signals in I, and it generates a truth assignment to the signals in O. Thus, with every sequence of inputs, the transducer associates a sequence of outputs, and it realizes ψ if all the computations that are generated by the interaction satisfy ψ.

We consider the synthesis problem for *latticed*-LTL (LLTL, for short) [37]. LLTL is an extension of LTL to a multi-valued domain comprised of a lattice $\mathcal{L}$. A lattice is a partially-ordered set $\mathcal{L} = \langle A, \leqslant \rangle$ in which every two elements ℓ and ℓ' have a least upper bound (ℓ *join* ℓ') and a greatest lower bound (ℓ *meet* ℓ').

Thus, an LLTL formula ψ maps computations in which the atomic propositions take values from a lattice $\mathcal{L}$, to a satisfaction value in $\mathcal{L}$. We demonstrate this setting with an example.

Consider a setting in which three agents a, b, and c have different view-points on the atomic propositions AP in a system S. A truth assignment for the atomic propositions is then a function in $(2^{\{a,b,c\}})^{AP}$ assigning to each $p \in AP$ the set of agents according to whose view-point p is true. We reason about S using the lattice $\mathcal{L} = \langle 2^{\{a,b,c\}}, \subseteq \rangle$. For example, the truth value of the formula $\psi = \mathsf{G}(req \rightarrow \mathsf{F}grant)$ in a computation is the set of agents according to whose view-point, whenever a request is sent, it is eventually granted.

For the classical synthesis problem, researchers have studied a setting with incomplete information, where the truth values of some of the input signals are hidden or may contain errors, and the transducer should nevertheless realize ψ. In this work we introduce and study a model of incomplete information in the multi-valued setting. In our model, the system always reads all input signals, but their value may be perturbed according to a known noise function. This setting naturally models incomplete information in real-life multi-valued settings. For example, when the input is read by sensors that are not accurate (e.g., due to bounded precision, or to probabilistic measuring) or when the input is received over a noisy channel and may come with some distortion.

We study theoretical properties of LLTL: we start with cases where the set of attainable truth values of an LLTL formula are closed under join, thus a maximal attainable value exists, even when the lattice elements are partially ordered. We proceed to study stability properties, namely the affect of perturbing the values of the atomic propositions on the satisfaction value of formulas. Next, we continue to the synthesis and the noisy-synthesis problems for LLTL, which we solve via a translation of LLTL formulas to Boolean automata. We show that by working with universal automata, we can handle the exponential blow-up that incomplete information involves together with the exponential blow-up that determinization involves, thus the noisy-synthesis problem stays 2EXPTIME-complete, as it is for LTL.

These results have been described in [8], and appear in Chapter 3 of this thesis.

Repairing Multi-Player Games

LTL synthesis can be thought of as a two-player zero-sum game between the system and the environment, where the goal of the system is to satisfy the specification, and the environment is adversarial. Then, synthesis amounts to finding a winning strategy for the system.

Modern systems, however, often consist of several interacting components, each having its own objective. The interaction among the components is modeled by a *multi-player game*. Each player in the game corresponds to a component in the interaction. In each round of the game, each of the players chooses an action, and the next vertex of the game depends on the current vertex and the vector of actions chosen. A strategy for a player is then a mapping from the history of the game so far to her next action.

The strategies of the players induce a trace in the game, and the goal of each player is to direct the game into a trace that satisfies her specification. This is modeled by augmenting the game with ω-regular winning conditions, describing the objectives of the players. Unlike traditional synthesis games, which are zero-sum, here the objectives of the players do not necessarily contradict each other. Accordingly, typical questions about these games concern their stability – whether the players reach a Nash Equilibrium (NE, for short), and their social welfare – maximizing the set of (possibly weighted) specifications that are satisfied [45]. It is easy to find instances of such games with no NE.

We introduce and study *repair* of multi-player games. We consider a setting with an authority (the designer) that aims to stabilize the interaction among the components and to increase the social welfare. In standard reactive synthesis [46], there are various ways to cope with situations when a specification is not realizable. Obviously, the specification has to be weakened, and this can be done either by adding assumptions on the behavior of the environment, fairness included, or by giving up some of the requirements on the system [19, 42]. In our setting, where the goal is to obtain stability, and the game is not zero-sum, a repair may both weaken and strengthen the specifications, which, in our main model, is modeled by modifications to the winning conditions.

The input to the *specification-repair problem* (SR problem, for short) is a game along with a *cost function*, describing the cost of each repair. For example, in Büchi games the cost function specifies, for each vertex v and player i, the cost of making v accepting for Player i and the cost of making v rejecting. The cost may be 0, reflecting the fact that v is accepting or rejecting in the original specification of Player i, or it may be ∞, reflecting the fact that the original classification of v is a feature of the specification that the designer is not allowed to modify. We consider some useful classes of cost functions, like *uniform costs* – where all assignments cost 1, except for one that has cost 0 and stands for the original classification of the vertex, or *don't-care costs* – where several assignments have cost 0, reflecting a don't-care original classification, and all other assignments have cost ∞.

The goal of the designer is to suggest a repair to the winning conditions with which the game has an NE. One way to quantify the quality of a repair is its cost, and indeed the problem also gets as input a bound on the budget that can be used in the repairs. Another way, which has to do with the social welfare, considers the specifications that are satisfied in the obtained NE. Specifically, in the *rewarded specification-repair problem* (RSR problem, for short), the input also includes a *reward function* that maps subsets of specifications to rewards. When the suggested repair leads to an NE with a set W of winners, the designer gets a reward that corresponds to the specifications of the players in W. The quality of a solution then refers both to the budget it requires and to its reward.

Studying the SR and RSR problems, we distinguish between several classes, characterized by the type of winning conditions, cost functions, and reward functions. From a complexity point of view, we also distinguish between the case where the number of players is arbitrary and the one where it is constant. The problem of deciding whether an NE exists is known to be NP-complete with an arbitrary number of players for most ω-regular objectives, but can be solved in polynomial time for Büchi games [16]. It is not too hard to lift the NP lower bound to the SR and RSR problems. The main challenge is the Büchi case, where one should find the cases where the polynomial complexity of deciding whether an NE exists can be lifted to the SR and RSR problems, and the cases where the need to find a repair shifts the complexity of the problem to NP. We show that the polynomial complexity can be maintained for don't-care costs, but the other settings are NP-complete. We then check whether fixing the number of players can reduce the complexity of the SR and RSR problems, either by analyzing the complexity of the algorithms for an arbitrary number of players, or by introducing new algorithms. We show that in many cases, we can solve the problem in polynomial time, mainly thanks to the fact that it is possible to go over all possible subsets of players in search for a subset that can win in an NE.

These results have been described in [2], and do not appear in the thesis due to lack of space.

Sensing

Studying the complexity of a formal language, there are several complexity measures to consider. When the language is given by means of a Turing Machine, the traditional measures are time and space demands. For regular and ω-regular languages, given by means of finite-state automata, the classical complexity measure is the size of a minimal deterministic automaton that recognizes the language.

We introduce and study a new complexity measure, namely the *sensing cost* of the language. Intuitively, the sensing cost of a language measures the detail with which a random input word needs to be read in order to decide membership in the language. The sensing cost is a quality measure in which the source of quantitativeness is the system, and is independent of the specification. For example, System C in Example 1.1 has a lower sensing cost than System A, as C reads only one input every 100 steps. Our goal is to formalize regular sensing in the finite-state setting and to study the sensing complexity measure for regular and ω-regular languages.

We consider languages over alphabets of the form 2^P, for a finite set P of signals.

Consider a deterministic automaton $\mathcal{A}$ over an alphabet 2^P. For a state q of $\mathcal{A}$, we say that a signal $p \in P$ is *sensed* in q if at least one transition taken from q depends on the truth value of p. The *sensing cost* of q is the number of signals it senses, and the sensing cost of a run is the average sensing cost of states visited along the run. We extend the definition to automata by assuming a uniform distribution of the inputs. Thus, the sensing cost of $\mathcal{A}$ is the limit of the expected sensing of runs over words of increasing length. Finally the sensing cost of a language L, of either finite or infinite words, is then the infimum of the sensing costs of deterministic automata for L.

We study sensing in the context of regular and ω-regular languages, followed by a studies in the context of monitoring and synthesis of safety properties.

Regular Sensing

Our goal here is to compute the sensing cost of a language L. We start by showing that our definition of the sensing cost of an automaton $\mathcal{A}$ coincides with one that is based on the stationary distribution of a Markov chain induced by $\mathcal{A}$, which enables us to calculate the sensing cost of an automaton in polynomial time.

We start by studying the sensing cost of regular languages of finite words. For the complexity measure of size, the picture in the setting of finite words is very clean: each language L has a unique minimal deterministic automaton (DFA), namely the *residual automaton* $\mathcal{R}_L$ whose states correspond to the equivalence classes of the Myhill-Nerode right-congruence relation for L.

We show that minimizing the state space of a DFA can only reduce its sensing cost. Hence, the clean picture of the size measure is carried over to the sensing measure: the sensing cost of a language L is attained in the DFA $\mathcal{R}_L$. In particular, since DFAs can be minimized in polynomial time, we can construct in polynomial time a minimally-sensing DFA, and can compute in polynomial time the sensing cost of languages given by DFAs.

We then study the sensing cost of ω-regular languages, given by means of deterministic parity automata (DPAs). Recall the size complexity measure. There, the picture for languages of infinite words is not clean: A language needs not have a unique minimal DPA, and the problem of finding one is NP-complete [47]. It turns out that the situation is challenging also in the sensing measure. First, we show that different minimal DPAs for a language may have different sensing costs. In fact, bigger DPAs may have smaller sensing costs. This intricacy is demonstrated e.g. by System C in Example 1.1.

Our main result is that despite the above intricacy, the sensing cost of an ω-regular language L is the sensing cost of the residual automaton $\mathcal{R}_L$ for L. It follows that the sensing cost of an ω-regular language can be computed in polynomial time. Unlike the case of finite words, it may not be possible to define L on top of $\mathcal{R}_L$. Interestingly, however, $\mathcal{R}_L$ does capture exactly the sensing required for recognizing L. The proof of this property of $\mathcal{R}_L$ is the main technical challenge of our contribution. The proof goes via a sequence $(\mathcal{B}_n)_{n=1}^{\infty}$ of DPAs whose sensing costs converge to that of L. The DPA $\mathcal{B}_n$ is obtained from a DPA $\mathcal{A}$ for L by a lazy sensing strategy that spends time in n copies of $\mathcal{R}_L$ between visits to $\mathcal{A}$, but spends enough time in $\mathcal{A}$ to ensure that the language is L.

These results have been described in [9], and appear in Chapter 4 of this thesis.

The Sensing Cost of Monitoring Safety Properties

An appealing application for sensing is *monitoring*: we are given a computation and we have to decide whether it satisfies a specification. Monitoring is especially useful when reasoning about *safety* specifications [32]. There, every computation that violates the specification has a bad prefix – one all whose extensions are not in L. Hence, as long as the computation is a prefix of some word in L, the monitor continues to sense and examine the computation. Once a bad prefix is detected, the monitor declares an error and no further sensing is required.

The definition of sensing cost above falls short in monitoring safety properties. Indeed, the definition above does not distinguish between words in the language and words not in the language, whereas in monitoring we care only for words in the language. In particular, according to the definition above, the sensing cost of a safety language is always 0.

Thus, in order to define a meaningful sensing measure for safety properties, we need to consider, instead of the uniform distribution above, more intricate probability spaces – ones that restrict attention to words in the language. As we show, there is more than one way to define such probability spaces, each leading to a different measure. We study two such measures. In the first, we sample a word randomly, letter by letter, according to a given distribution, allowing only letters that do not generate bad prefixes. In the second, we construct a sample space directly on the words in the language. We show that in both definitions, we can compute the sensing cost of the language in polynomial time, and that the minimal sensing cost is attained by a minimal-size automaton. Thus, luckily enough, even though different ways in which a computation may be given in an online manner calls for two definitions of sensing cost, the design of a minimally-sensing monitor is the same in the two definitions.

These results have been described in [10], and appear in Chapter 5 of this thesis.

The Sensing Cost of Synthesis

A natural setting in which sensing arises is *synthesis*: given a specification over sets I and O of input and output signals, the goal is to construct a finite-state system that, given a sequence of input signals, generates a computation that satisfies the specification. In each moment in time, the system reads an assignment to the input signals, namely a letter in

2^I, which requires the activation of $|I|$ Boolean sensors. A well-studied special case of limited sensing is synthesis with *incomplete information*. There, the system can read only a subset of the signals in I, and should still generate only computations that satisfy the specification [38, 20]. A more sophisticated case of sensing in the context of synthesis is studied in [21], where the system can read some of the input signals some of the time. In more detail, sensing the truth value of an input signal has a cost, the system has a budget for sensing, and it tries to realize the specification while minimizing the required sensing budget. In this work, we restrict our specifications to safety properties.

The definition of sensing above considers automata and does not partition P into I and O, whereas synthesis refers to I/O-transducers. Moreover, unlike automata, correct transducers generate only computations in the language, and they need not generate all words in the language – only these that ensure receptiveness with respect to all sequences of inputs. The latter is the main challenge in finding a minimally-sensing transducer for a specification.

Giving up sensing has a flavor of synthesis with incomplete information [39]: the transducer has to realize the specification no matter what the incomplete information is. This introduces a new degree of freedom, which requires different techniques than those used above. In particular, while a minimal-size transducer for a safety language can be defined on top of the state space of a minimal-size deterministic automaton for the language, this is not the case when we seek minimally-sensing transducers. This is different also from the results in the monitoring setting, where a minimally-sensing automaton or monitor for a safety language coincides with the minimal-size automaton for it. In fact, we show that a minimally-sensing transducer for a safety language might be exponentially bigger than a minimal-size automaton for the language. Consequently, the problems of computing the minimal sensing cost and finding a minimally-sensing transducer are EXPTIME-complete even for specifications given by means of deterministic automata. On the positive side, a transducer that attains the minimal sensing cost always exists.

These results have also been described in [10], and appear in Chapter 5 of this thesis.

Weighted Automata

As can be seen in several works above, automata play a major role in formal methods, both as specification formalisms, and as technical tools. Indeed, The *automata-theoretic approach* uses the theory of automata as a unifying paradigm for system specification, verification, and synthesis [40, 51]. By viewing computations as words (over the alphabet of possible assignments to the variables of the system), we can view both the system and the specification as languages. Then, problems such as model-checking, satisfiability, etc. reduce to questions about automata and their languages

Traditional automata accept or reject their input, and are therefore Boolean. This stands in strong relation to the fact that traditional logics are Boolean. The study of quantitative properties led researchers to study *weighted automata* [27]. A weighted automaton maps each word to a value, which may come from a rich domain (e.g. $\mathbb{N}$, a lattice, etc.).

While there are many models of weighted automata, one of the most prominent models is that of weighted finite automata under the *tropical semiring* $\langle \mathbb{R}^{\geqslant 0} \cup \{\infty\}, \min, +, \infty, 0\rangle$ (WFAs, for short). WFAs run on finite words, similarly to NFAs. Each transition of a WFA is assigned a cost in $\mathbb{R}^{\geqslant 0}$, and the cost of a run is the sum of the costs of the transitions taken along the run. The cost $\mathcal{A}(w)$ of a word w is then the minimal cost of a run of the WFA w.

Applications of WFAs include formal verification, where they are used for the verification of quantitative properties [17, 18, 26] for reasoning about probabilistic systems [13], and for reasoning about the competitive ratio of on-line algorithms [11], as well as text, speech, and image processing, where the costs of the automaton are used in order to account for the variability of the data and to rank alternative hypotheses.

The Universality Problem for WFAs

The rich structure of WFAs makes them intriguing mathematical objects. Fundamental problems that have been solved decades ago for non-weighted automata are still open or known to be undecidable in the weighted setting.

We study of the universality problem for WFAs: given a weighted automaton $\mathcal{A}$, whether $\mathcal{A}(w) < 0$ for every word w. This problem was shown to be undecidable in [34]. We describe a new proof of this result. Our clean reduction enables us to strengthen the result to a weaker model of automata, and to make the proof generalizable to automata over infinite words. In addition, we show that if all the weights in the automaton are non-negative, then the universality problem becomes PSPACE-complete.

These results have been described in [3], and do not appear in the thesis due to lack of space.

Alternating Weighted Automata

The second context in which we study WFAs is that of alternation. In the Boolean setting, the model of *alternating* automata has proven to be especially useful in the context of formal verification. While in a nondeterministic automaton the transition function specifies only existential requirements on the run, in an alternating automaton it specifies both existential and universal requirements. The universal requirements correspond to conjunctions in the specifications, making the translation of temporal-logic formulas to alternating automata simple and linear [30, 49], as opposed to the exponential translation to nondeterministic automata [51]. The linear translation of temporal logic to alternating automata is essential in automata-based algorithms for model checking of branching temporal logic formulas [40], and is useful for further minimization of the automata [48], for handling of incomplete information [39], for algorithms that avoid determinization [35], and more.

In the Boolean setting, the semantics of both disjunctions and conjunctions is straightforward. Recall that in a nondeterministic weighted automaton, the weight of a word is the value of the cheapest run on it. This meets our intuition of a "minimum semantics" for disjunctions in the weighted setting. It is less clear what the semantics of conjunctions should be. If we want to maintain the traditional helpful dualization between disjunctions and conjunctions, then an appropriate semantics for conjunction is a *maximum* semantics. The maximum semantics is also suitable for an analysis in which the weights correspond to a confidence or a truth-level indication. However, if the analysis is used in order to study the *cost* of the satisfaction, then an appropriate semantics is a *summation* semantics, in which the cost of satisfying a conjunction $\varphi_1 \wedge \varphi_2$ is not the maximal cost of satisfying φ_1 or φ_2, but rather the sum of these costs. Note that for the motivation of quantitative specifications, where one wants to replace Boolean satisfaction by a quantitative value that describes the quality of the satisfaction, both the maximum and the summation semantics are of interest. The two possible semantics for conjunctions in the weighted setting induce two different semantics for universal branches in alternating weighted automata.

We study alternating weighted automata on finite words, in both the maximum and summation semantics (Alternating automata with the maximum semantics were studied in [18] over infinite words, which pose different challenges). We refer to the automata by MAX-AWAs and SUM-AWAs, respectively. We start with the max semantics. We study the expressive power of MAX-AWAs, their closure properties with respect to the operators min, max, and negation of weighted languages, the power of alternation with respect to nondeterminism, and arithmetics with MAX-AWAs. We also formalize the duality between the min semantics of existential transitions and the max semantics of universal transitions by means of a negation operator on weighted languages.

We continue and study the sum semantics. A key difference between MAX-AWAs and SUM-AWAs is the fact that in the sum semantics the weight of a word may be exponentially larger than its length (even when all costs in the automaton are bounded by a constant). One immediate implication of this is that alternation cannot be removed in SUM-AWA. We also study closure properties for SUM-AWA, and the added expressive power of alternation even in languages in which the weight of a word does not go beyond its length.

These results have been described in [7], and do not appear in the thesis due to lack of space.

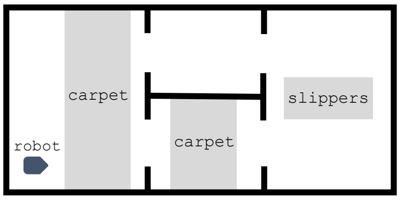


Figure 1: A mobile robot in a room with the task specification, "do not go on the carpets until the slippers are reached."

Robotic Planning with Partial Satisfaction

The last work we present is of a more practical nature, and concerns robotic planning. The specification of complex motion goals through LTL is increasingly favored in robotics to narrow the gap between task and motion planning [23]. The power of these methods lie in the expressiveness of LTL and the correctness and completeness guarantees provided by its synthesis algorithms [14, 15, 25, 52].

A major barrier of utilizing existing LTL planners in realistic environments is their Boolean satisfaction condition. These frameworks output a plan only if the plan fully satisfies the specification. Even though generating a satisfying plan is ideal, there are scenarios where the full realization of a task may not be possible; nevertheless, parts of the specification may still be achievable and of interest to the user. For example, consider the robotic scenario depicted in Figure 1 with the user specification, "do not go on the carpet until the slippers are reached." Clearly, this specification is unsatisfiable as the robot must go on the carpet first to reach the slippers. Now imagine that reaching slippers is of a higher priority to the user than avoiding carpets. With this information, the desired robot behavior is obviously to reach the slippers by violating the carpet constraint only once. As robots enter unstructured environments, such scenarios become more common. This work introduces a method for *quantification of satisfaction* of an LTL specification and proposes a planner that handles such scenarios, via weighted automata.

The approach views atomic propositions in an LTL specification as tasks, for which the user can assign priorities. These proposition priorities are then translated to costs of violations of the specification. These costs define a distance between system trajectories with respect to the satisfaction of the specification. Given a robotic system, a specification in a fragment of LTL, and costs over propositions, the planning framework automatically synthesizes a continuous plan with the minimum distance to satisfaction of the specification.

Technically, the planner first constructs a weighted automaton by combining the proposition costs with the automaton that represents the specification. Thus, the notion of distance to satisfaction derives from the automaton that represents the specification. Next, the framework computes a discrete abstraction for the continuous system and composes the abstraction with the weighted automaton. The result is a weighted graph, which is employed to generate high-level plans in the multi-layered planning framework in [43]. These high-level plans, through a synergy layer, guide a low-level planner to explore the state space of the robotic system to generate a system trajectory with the minimum distance to satisfaction.

These results have been described in [41], and do not appear in the thesis due to lack of space.

References

[1] S. Almagor, G. Avni, and O. Kupferman. Automatic generation of quality specifications. In *25th CAV*, volume 8044 of *Lecture Notes in Computer Science*, pages 479–494, 2013.

[2] S. Almagor, G. Avni, and O. Kupferman. Repairing multi-player games. In *26th International Conference on Concurrency Theory, CONCUR 2015, Madrid, Spain, September 1.4, 2015*, pages 325–339, 2015.

[3] S. Almagor, U. Boker, and O. Kupferman. What's decidable about weighted automata? In *9th Int. Symp. on Automated Technology for Verification and Analysis*, volume 6996 of *Lecture Notes in Computer Science*, pages 482–491. Springer, 2011.

[4] S. Almagor, U. Boker, and O. Kupferman. Formalizing and reasoning about quality. In *Proc. 40th Int. Colloq. on Automata, Languages, and Programming*, volume 7966 of *Lecture Notes in Computer Science*, pages 15 – 27. Springer, 2013.

[5] S. Almagor, U. Boker, and O. Kupferman. Discounting in LTL. In *Proc. 20th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science. Springer, 2014.

[6] S. Almagor, U. Boker, and O. Kupferman. Formally Reasoning About Quality. In *Journal of the ACM*, Accepted for publication, 2016.

[7] S. Almagor and O. Kupferman. Max and sum semantics for alternating weighted automata. In *9th Int. Symp. on Automated Technology for Verification and Analysis*, volume 6996 of *Lecture Notes in Computer Science*, pages 13–27. Springer, 2011.

[8] S. Almagor and O. Kupferman. Latticed-ltl synthesis in the presence of noisy inputs. In *Proc. 17th Int. Conf. on Foundations of Software Science and Computation Structures*, Lecture Notes in Computer Science. Springer, 2014.

[9] S. Almagor, D. Kuperberg, and O. Kupferman. Regular sensing. In *34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2014, December 15-17, 2014, New Delhi, India*, pages 161–173, 2014.

[10] S. Almagor, D. Kuperberg, and O. Kupferman. The Sensing Cost of Monitoring and Synthesis. In *35th International Conference on Foundation of Software Technology and Theoretical Computer Science*, pages 380–393, 2015.

[11] B. Aminof, O. Kupferman, and R. Lampert. Reasoning about online algorithms with weighted automata. *ACM Transactions on Algorithms*, 6(2), 2010.

[12] D. Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75(2):87–106, 1987.

[13] C. Baier, N. Bertrand, and M Grösser. Probabilistic automata over infinite words: Expressiveness, efficiency, and decidability. In *Proc. 11th International Workshop on Descriptional Complexity of Formal Systems*, pages 3 – 16, 2006.

[14] A. Bhatia, L. E. Kavraki, and M. Y. Vardi. Motion planning with hybrid dynamics and temporal goals. In *IEEE Conf. on Decision and Control*, 1108–1115, 2010.

[15] A. Bhatia, L. E. Kavraki, and M. Y. Vardi. Sampling-based motion planning with temporal goals. In *Int. Conf. on Robotics and Automation*, 2689–2696, 2010.

[16] P. Bouyer, R. Brenguier, and N. Markey. Nash equilibria for reachability objectives in multi-player timed games. In *Proc. 21st CONCUR*, pages 192–206, 2010.

[17] K. Chatterjee, L. Doyen, and T. Henzinger. Quantative languages. In *Proc. 17th Annual Conf. of the European Association for Computer Science Logic*, pages 385–400, 2008.

[18] K. Chatterjee, L. Doyen, and T. Henzinger. Alternating weighted automata. In *Proc. 17th International Symposium on Fundamentals of Computation Theory*, volume 5699, pages 3–13, 2009.

[19] K. Chatterjee, T. Henzinger, and B. Jobstmann. Environment assumptions for synthesis. In *Proc. 19th Int. Conf. on Concurrency Theory*, volume 5201 of *Lecture Notes in Computer Science*, pages 147–161. Springer, 2008.

[20] K. Chatterjee and R. Majumdar. Minimum attention controller synthesis for omega-regular objectives. In *FORMATS*, pages 145–159, 2011.

[21] K. Chatterjee, R. Majumdar, and T. A. Henzinger. Controller synthesis with budget constraints. In *Proc 11th International Workshop on Hybrid Systems: Computation and Control*, volume 4981 of *Lecture Notes in Computer Science*, pages 72–86. Springer, 2008.

[22] E.M. Clarke and E.A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proc. Workshop on Logic of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer, 1981.

[23] E. M. Clarke, O. Grumberg, and D. Peled. *Model checking*. MIT Press, 1999.

[24] L. de Alfaro, M. Faella, T.A. Henzinger, R. Majumdar, and M. Stoelinga. Model checking discounted temporal properties. *Theoretical Computer Science*, 345(1):139–170, 2005.

[25] J. A. DeCastro and H. Kress-Gazit. Guaranteeing reactive high-level behaviors for robots with complex dynamics. In *Int. Conf. on Intelligent Robots and Systems*, 749–756, 2013.

[26] M. Droste and P. Gastin. Weighted automata and weighted logics. In *Proc. 32nd Int. Colloq. on Automata, Languages, and Programming*, pages 513–525, 2005.

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

[28] E.M. Clarke and E.A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In Proc. Workshop on Logic of Programs, volume 131 of Lecture Notes in Computer Science, pages 5271. Springer, 1981.

[29] C. Eisner and D. Fisman. A Practical Introduction to PSL. Springer, 2006.

[30] E.A. Emerson and C. Jutla. Tree automata, μ-calculus and determinacy. In *Proc. 32nd IEEE Symp. on Foundations of Computer Science*, pages 368–377, 1991.

[31] G.J. Holzmann. The Model Checker SPIN. IEEE Transactions on Software Engineering, 23(5):279295, 1997.

[32] K. Havelund and G. Rosu. Efficient monitoring of safety properties. *Software Tools for Technology Transfer*, 6(2):18–173, 2004.

[33] B. Jobstmann, S. Galler, M. Weiglhofer, and R. Bloem. Anzu: A tool for property synthesis. In *Proc. 19th Int. Conf. on Computer Aided Verification*, volume 4590 of *Lecture Notes in Computer Science*, pages 258–262, 2007.

[34] D. Krob. The equality problem for rational series with multiplicities in the tropical semiring is undecidable. *International Journal of Algebra and Computation*, 4(3):405–425, 1994.

[35] O. Kupferman. Avoiding determinization. In *Proc. 21st IEEE Symp. on Logic in Computer Science*, pages 243–254, 2006.

[36] R.P. Kurshan. FormalCheck User's Manual. Cadence Design, Inc., 1998.

[37] O. Kupferman and Y. Lustig. Lattice automata. In *Proc. 8th Int. Conf. on Verification, Model Checking, and Abstract Interpretation*, volume 4349 of *Lecture Notes in Computer Science*, pages 199 – 213. Springer, 2007.

[38] O. Kupferman and M.Y. Vardi. Church's problem revisited. *The Bulletin of Symbolic Logic*, 5(2):245 – 263, 1999.

[39] O. Kupferman and M.Y. Vardi. Synthesis with incomplete information. In *Advances in Temporal Logic*, pages 109–127. Kluwer Academic Publishers, 2000.

[40] O. Kupferman, M.Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *Journal of the ACM*, 47(2):312–360, 2000.

[41] M. Lahijanian, S. Almagor, D Fried, L.E. Kavraki, and M.Y. Vardi. This Time the Robot Settles for a Cost: A Quantitative Approach to Temporal Logic Planning with Partial Satisfaction. In *29th AAAI*, pages 3664–3671, January 25-30, 2015, Austin, Texas, USA.

[42] W. Li, L. Dworkin, and S. A. Seshia. Mining assumptions for synthesis. In *Proc. 9th MEMOCODE*, pages 43–50, 2011.

[43] M. R.Maly, M. Lahijanian, L. E. Kavraki, H. Kress-Gazit, and M. Y. Vardi. Iterative temporal motion planning for hybrid systems in partially unknown environments. In *Int. Conf. on Hybrid Systems: Computation and Control*, 353–362, 2013.

[44] M.L. Minsky. *Computation: Finite and Infinite Machines*. Prentice Hall, 1 edition, 1967.

[45] J. von Neumann and O. Morgenstern. Theory of games and economic behavior.

[46] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. 16th ACM Symp. on Principles of Programming Languages*, pages 179–190, 1989.

[47] S. Schewe. Beyond Hyper-Minimisation—Minimising DBAs and DPAs is NP-Complete. volume 8 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 400–411, 2010.

[48] F. Somenzi and R. Bloem. Efficient Büchi automata from LTL formulae. In *Proc. 12th Int. Conf. on Computer Aided Verification*, volume 1855 of *Lecture Notes in Computer Science*, pages 248–263. Springer, 2000.

[49] M.Y. Vardi. Nontraditional applications of automata theory. In *Proc. 11th Symp. on Theoretical Aspects of Computer Science*, volume 789 of *Lecture Notes in Computer Science*, pages 575–597. Springer, 1994.

[50] M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. 1st IEEE Symp. on Logic in Computer Science*, pages 332–344, 1986.

[51] M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.

[52] C. Vasile and C. Belta Sampling-based temporal logic path planning. In *Int. Conf. on Intelligent Robots and Systems*, 4817–4822, 2013.

Formally Reasoning About Quality*

Shaull Almagor[†] *Udi Boker* [‡] *Orna Kupferman* [§]

Abstract

In recent years, there is growing need and interest in formally reasoning about the quality of software and hardware systems. As opposed to traditional verification, where one considers the question of whether a system satisfies, or not, a given specification, 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 paper 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.

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 [20]. 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

*Accepted for publication in the JACM. The present article combines and extends [4] and [5].

[†] The Hebrew University, Jerusalem, Israel.

[‡] The Interdisciplinary Center, Herzliya, Israel.

[§] The Hebrew University, Jerusalem, Israel.

quantitative aspects (multi-valued / probabilistic / fuzzy) [62, 54, 35, 30, 31], but is applied also with respect to Boolean systems, where it origins from the semantics of the specification formalism itself [22, 4].

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 $\psi = \mathsf{G}(request \rightarrow \mathsf{F}(response\_grant \vee response\_deny))$; that is, every request is eventually responded, 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 [43, 68]. Quality, however, is traditionally viewed as an art, or as an amorphous ideal. In this paper 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 he gives to components such as security, maintainability, runtime, delays, and more, and then can formally reason about the quality of hardware and software systems.

1.1 Our Contribution

Recall the specification example $\psi = \mathsf{G}(request \rightarrow \mathsf{F}(response\_grant \vee response\_deny))$ above. As the specification demonstrates, one can distinguish between two aspects of the quality of satisfaction. The first, to which we refer as "propositional quality", concerns prioritizing related components of the specification. The second, to which we refer as "temporal quality", concerns the waiting time to satisfaction of eventualities. We study both aspects, as well as their combination. Below 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$[\mathcal{F}]$, which 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 [35]. 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 sub-specification. 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 $\mathsf{G}(req \rightarrow (grant \oplus_{\frac{3}{4}} \mathsf{X} grant))$. 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 [65]. 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 [10, 18]. 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 [71]. In the quantitative approach, it seems natural to translate formulas to *weighted automata* [61, 28]. However, these extensively-studied models are complicated and many problems become undecidable for them (e.g., the universality problem – [45, 3]). We show that we can use the approach taken in [35], 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

[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.

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 [71] 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 the "near" and "far" future includes bounded (prompt) eventualities [12, 6]. 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, and thus has 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 to augment temporal logics with*future discounting* [24]. In the discounted setting, the satisfaction value of specifications is a numerical value, and it depends, according to some discounting function, on the time waited for eventualities to get satisfied.

We introduce and study the linear temporal logic LTL$^{\text{disc}}[\mathcal{D}]$ – an augmentation by discounting of LTL. The logic LTL$^{\text{disc}}[\mathcal{D}]$ 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). $\text{LTL}^{\text{disc}}[\mathcal{D}]$ includes a discounting-"until" (U_η) operator, parameterized by a function $\eta \in \mathcal{D}$. We solve the model-checking threshold problem for $\text{LTL}^{\text{disc}}[\mathcal{D}]$: given a Kripke structure $\mathcal{K}$, an $\text{LTL}^{\text{disc}}[\mathcal{D}]$ 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.

Similarly to the case of $\text{LTL}[\mathcal{F}]$, an attempt to handle $\text{LTL}^{\text{disc}}[\mathcal{D}]$ with weighted automata involves problems that are in general undecidable [3]. Unlike $\text{LTL}[\mathcal{F}]$, the range of possible satisfaction values of an $\text{LTL}^{\text{disc}}[\mathcal{D}]$ formula is infinite. Consequently, the border of decidability is different and the use of Boolean automata in reasoning about $\text{LTL}^{\text{disc}}[\mathcal{D}]$ is much more challenging. Nevertheless, we show that for threshold problems, we can translate $\text{LTL}^{\text{disc}}[\mathcal{D}]$ formulas into (Boolean) nondeterministic Büchi automata (NBA), with the property that the automaton accepts a lasso computation iff the formula attains a value above the threshold on that computation. Our algorithm relies on the fact that the language of an automaton is non-empty iff there is a lasso witness for the non-emptiness. 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 $\text{LTL}^{\text{disc}}[\mathcal{D}]$ with standard discounting functions, such as exponential decaying (denoted by $\text{LTL}^{\text{disc}}[E]$), 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 $\text{LTL}[\mathcal{F}]$, the fact that in $\text{LTL}^{\text{disc}}[\mathcal{D}]$ there are infinitely many satisfaction values implies that solving the threshold decision problems does not yield a solution to the search problem. Indeed, 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 $\text{LTL}[\mathcal{F}]$ and $\text{LTL}^{\text{disc}}[\mathcal{D}]$ and studying their decision problems, we augment $\text{LTL}^{\text{disc}}[\mathcal{D}]$ with propositional quality operators. Beyond the operators $\min$, $\max$, and $\neg$, which are already present, two basic propositional quality operators of $\text{LTL}[\mathcal{F}]$ are the multiplication of an $\text{LTL}^{\text{disc}}[\mathcal{D}]$ formula by a constant in $[0, 1]$, and the averaging between the satisfaction values of two $\text{LTL}^{\text{disc}}[\mathcal{D}]$ formulas. We show that while the first extension does not increase the expressive power of $\text{LTL}^{\text{disc}}[\mathcal{D}]$ 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, where the complexity of the model-checking problem stays polynomial [22].

We consider additional extensions of $\text{LTL}^{\text{disc}}[\mathcal{D}]$. 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 $\text{LTL}^{\text{disc}}[\mathcal{D}]$ *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 [56, 55]. We show that the same holds for the discounted setting. Finally, we show how $\text{LTL}^{\text{disc}}[\mathcal{D}]$ and algorithms for it can be used also 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 [47, 29, 30, 7, 31].

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 (multi-valued / 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 (for example, by associating a conjunction with *min*) [62, 35], 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 below on some of these works.

- Early work handles *multi-valued systems*, in which the values of atomic propositions are taken from a finite domain. For example, "uninitialized", "unknown", and "don't care" [42]. Beyond richer expressiveness, multi values, sometimes arranged in a lattice, are useful in abstraction methods, query checking, and verification of systems from inconsistent viewpoints, where multi values are used to model missing, hidden, or varying information [17, 47]. Satisfaction of formulas is then multi-valued, corresponding to the values of atomic propositions.

- More sophisticated multi-valued systems consider *real-valued signals* [26]. The quantitative setting there stems from both time intervals and predicates over the value of atomic propositions. In particular, signal temporal logic [27] 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, e.g., [62], formulas are interpreted over fuzzy systems – ones where 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 been studied also in the context of quantitative verification [23, 35]. 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 [40, 25, 54]. 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 [67]. In the area of formal verification, it was suggested in [24] 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, [24] shows how to evaluate discounted μ-calculus formulas with arbitrary precision. Formulas of LTL can be translated to the μ-calculus, thus [24] 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 [21] (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 e.g., [13], in which LTL is augmented with Boolean assertions over the sum and average of quantitative atomic assertions, and [11], 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. In the more recent [16], the authors 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 [22], where CTL is augmented with discounting and weighted-average operators. The motivation in [22] 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 [57], where a version of discounted-LTL was introduced. Semantically, there are two main differences between the logics. The first is that [57] uses discounted sum, while we interpret discounting without accumulation. The second is that the discounting there replaces the standard temporal operators, so 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, and indeed the contribution in [57] is a comprehensive study of the mathematical properties of the logic. Yet, [57] does not study algorithmic questions about to the logic. We, on the other hand, focus on the algorithmic properties of the logic, and 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. In [37], the authors suggest an extension of $\text{LTL}^{\text{disc}}[E]$ to continuous-time computations. In the more applicative front, [63] had recently used our framework in order to solve the synthesis problem with a near-optimal quality. There, 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 our work in [2] we automatically generate the factors in $\text{LTL}^{\triangledown}$ formulas according to samples of computations and corresponding satisfaction values, and in [8] we study $\text{LTL}[\mathcal{F}]$ synthesis in a stochastic setting, where we maximize the expected satisfaction value of a synthesized transducer.

2 Formalizing Propositional Quality

In this section we introduce the temporal logic $\text{LTL}[\mathcal{F}]$ (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 $\text{LTL}[\mathcal{F}]$ (Section 2.3), followed by an algorithm for translating a given $\text{LTL}[\mathcal{F}]$ formula into an automaton (Section 2.4). This automata-theoretic approach will be our main tool in solving decision and search problems for $\text{LTL}[\mathcal{F}]$ (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 $p \in AP$.
- $f(\varphi_1, ..., \varphi_k)$, $\mathsf{X}\varphi_1$, 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 consider also 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 1 below.[2]

Formula	Satisfaction value
$[\![\pi, \texttt{True}]\!]$	1
$[\![\pi, \texttt{False}]\!]$	0
$[\![\pi, p]\!]$	1 if $p \in \pi_0$ 0 if $p \notin \pi_0$
$[\![\pi, f(\varphi_1, ..., \varphi_k)]\!]$	$f([\![\pi, \varphi_1]\!], ..., [\![\pi, \varphi_k]\!])$
$[\![\pi, \mathsf{X}\varphi_1]\!]$	$[\![\pi^1, \varphi_1]\!]$
$[\![\pi, \varphi_1 \mathsf{U} \varphi_2]\!]$	$\max\limits_{i \geqslant 0}\{\min\{[\![\pi^i, \varphi_2]\!], \min\limits_{0 \leqslant j < i} [\![\pi^j, \varphi_1]\!]\}\}$

Table 1: The semantics of LTL$[\mathcal{F}]$.

[2] The observant reader may be concerned by our use of $\max$ and $\min$ where $\sup$ and $\inf$ are in order. In Lemma 2.1 we prove that there are only finitely many satisfaction values for a formula φ, thus the semantics is well defined.

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 $i \geqslant 0$ that maximizes the minimum between the satisfaction value of φ_2 in π^i (that is, the satisfaction value of the eventuality) and all the satisfaction values of φ_1 in π^j for $0 \leqslant j < i$ (that is, 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 abbreviation for the corresponding functions, as described below. 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,

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

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 $\bigvee_{i \geqslant 0}([\![\pi^i, \varphi_2]\!] \wedge \bigwedge_{0 \leqslant j < i}[\![\pi^j, \varphi_1]\!])$, which coincides with the semantics of LTL.

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

- $\mathsf{F}\varphi_1 = \mathtt{True}\mathsf{U}\varphi_1$. Thus, $[\![\pi, \mathsf{F}\varphi_1]\!] = \max\limits_{i \geqslant 0}\{[\![\pi^i, \varphi_1]\!]\}$.
- $\mathsf{G}\varphi_1 = \neg\mathsf{F}\neg\varphi_1$. Thus, $[\![\pi, \mathsf{G}\varphi_1]\!] = \min\limits_{i \geqslant 0}\{[\![\pi^i, \varphi_1]\!]\}$.

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 $L : S \to 2^{AP}$ is a labeling function. A *trace* of $\mathcal{K}$ is a sequence $s = s_0, s_1, \ldots$ of states such that $s_0 \in I$ and for all $0 \leqslant j$, 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 $0 \leqslant j$.

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] $[\![\mathcal{K},\varphi]\!] = \min\{[\![\pi,\varphi]\!] : \pi \text{ is a computation of } \mathcal{K}\}$.

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 $j \geqslant 0$, 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 $j \geqslant 0$.

Examples. We demonstrate the usefulness of $\mathrm{LTL}[\mathcal{F}]$ with some examples. First, in Examples 2.1 and 2.2 we utilize the $\oplus$ and $\triangledown$ operators. Then, 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, and accordingly sending twice as many messages. Assume that the* $\mathrm{LTL}[\mathcal{F}]$ *formulas* φ_1 *and* φ_2 *specify the quality of each of the servers. The quality of the system is then specified by the* $\mathrm{LTL}[\mathcal{F}]$ *formula* $\varphi_1 \oplus_{\frac{2}{3}} \varphi_2$.

Example 2.2 *Consider a scheduler that receives requests and generates grants. Consider the* $\mathrm{LTL}[\mathcal{F}]$ *formula* $\mathsf{G}(req \to \mathsf{F}(grant \oplus_{\frac{1}{2}} \mathsf{X}grant)) \wedge \neg(\triangledown_{\frac{3}{4}} \mathsf{G}\neg req)$. *The satisfaction value of the formula is* 1 *if every request is eventually granted, and 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, where* 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* $\mathrm{LTL}[\mathcal{F}]$ *formula specifying the heat generated by the* i*-th 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* $\mathrm{LTL}[\mathcal{F}]$ *formulas or by weighted propositions*[4]. *Then, the* $\mathrm{LTL}[\mathcal{F}]$ *formula* $\mathsf{F}(f(\psi_1, ..., \psi_n))$ *specifies the highest heat of the room over the course of the servers' operation.*

[3] Since a Kripke structure may have infinitely many computations, here too we should have a-priori used $\inf$, and the use of $\min$ is justified by Lemma 2.1.

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

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* $\mathrm{LTL}[\mathcal{F}]$ *formula specifying the configuration of the i-th joint. As in the previous example, ψ_i need not be Boolean (it may be, for example, its angle, or it may be a nested* $\mathrm{LTL}[\mathcal{F}]$ *formula, like $retracted \oplus_{\frac{5}{6}} \mathsf{X} retracted$, indicating that ideally the joint is retracted for two time units, with the first time unit being much more important). The* $\mathrm{LTL}[\mathcal{F}]$ *formula $\mathsf{G}(grab \rightarrow f(\psi_1, \psi_2, ..., \psi_k))$ 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.*

2.2 The Search and Decision Questions

In the Boolean setting, an LTL formula maps computations to $\{\mathtt{True}, \mathtt{False}\}$. In the quantitative setting, an $\mathrm{LTL}[\mathcal{F}]$ 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. Below we specify these questions with respect to $\mathrm{LTL}[\mathcal{F}]$. While the definition here focuses on $\mathrm{LTL}[\mathcal{F}]$, 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 $\mathrm{LTL}[\mathcal{F}]$ formula φ and returns $\sup \{ [\![\pi, \varphi]\!] : \pi \text{ is a computation} \}$. Dually, the *validity* problem returns, given an $\mathrm{LTL}[\mathcal{F}]$ 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 $\mathrm{LTL}[\mathcal{F}]$, Lemma 2.1 guarantees that there are only finitely many possible satisfaction values for φ. Accordingly, we can replace the $\sup$ and $\inf$ operators above 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 $\mathrm{LTL}[\mathcal{F}]$ 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

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

- **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 above questions are search and optimization problems. It is sometimes interesting to consider the decision problems 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 $[\![\mathcal{K},\varphi]\!] \geqslant t$. For some problems, there are natural thresholds to consider. For example, in the implication problem, asking whether $\max\left\{[\![\pi,\varphi_1]\!] - [\![\pi,\varphi_2]\!] : \pi \text{ is a computation}\right\} \geqslant 0$ amounts to asking whether for all computations π, we have that $[\![\pi,\varphi_1]\!] \geqslant [\![\pi,\varphi_2]\!]$, which indeed captures implication.

Working with strict vs. non-strict 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.

[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.

Lemma 2.1 *For every* LTL$[\mathcal{F}]$ *formula* φ*, we have that* $|V(\varphi)| \leqslant 2^{|\varphi|}$.

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 $|V(\varphi)| \leqslant 2$. For the induction step, we consider the following cases.

- Let $\varphi = f(\psi_1, ..., \psi_k)$, 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 $1 \leqslant i \leqslant k$, we have that $|V(\psi_i)| \leqslant 2^{|\psi_i|}$. Thus, the number of possible inputs for f is at most $\prod_{i=1}^{k} 2^{|\psi_i|} = 2^{\sum_{i=1}^{k} |\psi_i|} \leqslant 2^{|\varphi|-1} < 2^{|\varphi|}$.
- 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 $|V(\varphi)| \leqslant |V(\psi_1)| + |V(\psi_2)|$. By the induction hypothesis, the latter is at most $2^{|\psi_1|} + 2^{|\psi_2|} \leqslant 2^{|\psi_1|+|\psi_2|} = 2^{|\varphi|-1} < 2^{|\varphi|}$.

□

The good news that follows from Lemma 2.1 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. As bad news, Lemma 2.1 only gives an exponential bound to the number of satisfaction values. We now show that this exponential bound is tight.

Example 2.5 *Consider the logic* LTL$[\{\oplus\}]$*, augmenting* LTL *with the average function, where for every* $x, y \in [0, 1]$ *we have* $x \oplus y = \frac{1}{2}x + \frac{1}{2}y$*. Let* $n \in \mathbb{N}$ *and consider the formula* $\varphi_n = p_1 \oplus (p_2 \oplus (p_3 \oplus (p_4 \oplus ...p_n))...)$*. 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*

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

Hence, every assignment $\pi_0 \subseteq \{p_1, ..., p_{n-1}\}$ *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 2.1 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 2.1. 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 $\psi = f(\varphi_1, ..., \varphi_k)$ then $v = f(v(\varphi_1), ..., v(\varphi_k))$.
- 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 (similarly to Lemma 2.1) 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 become 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 $\varphi_1 \geqslant \varphi_2$ or $\varphi_1 \geqslant v$, 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 2.2 *For every* $\mathrm{LTL}[\mathcal{F}]$ *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* $[\![\pi, \varphi]\!] \in P$ *iff* $\pi \models Bool(\varphi, P)$*.*

Proof: The proof is by induction on the structure of φ.

- $Bool(\mathtt{True}, P) = \begin{cases} \mathtt{True} & \text{if } 1 \in P \\ \mathtt{False} & \text{if } 1 \notin P. \end{cases}$

- $Bool(\mathtt{False}, P) = \begin{cases} \mathtt{True} & \text{if } 0 \in P \\ \mathtt{False} & \text{if } 0 \notin P. \end{cases}$

- For $p \in AP$, we have $Bool(p, P) = \begin{cases} \mathtt{True} & \text{if } 0 \in P \text{ and } 1 \in P \\ p & \text{if } 0 \notin P \text{ and } 1 \in P \\ \neg p & \text{if } 0 \in P \text{ and } 1 \notin P \\ \mathtt{False} & \text{if } 0 \notin P \text{ and } 1 \notin P. \end{cases}$

- $Bool(f(\varphi_1, ..., \varphi_k), P) =$
$\bigvee_{\{d_1 \in V(\varphi_1), ..., d_k \in V(\varphi_k) :\, f(d_1, ..., d_k) \in P\}} Bool(\varphi_1, d_1) \wedge \cdots \wedge Bool(\varphi_k, d_k)$.

- $Bool(X\varphi_1, P) = \mathsf{X}Bool(\varphi_1, P)$.

- 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 $[\![\pi, \varphi]\!] \geqslant c$, and the second part guarantees that $[\![\pi, \varphi]\!] \not> c$.

We note that in the construction of $Bool(f(\varphi_1, ..., \varphi_k), P)$ 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 2.2 may involve an exponential blow-up. It is thus interesting to study whether this blow-up is unavoidable. One needs to tread carefully when studying the blow-up, 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 2.2, 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 $\mathrm{LTL}[\mathcal{F}]$, we can represent every Boolean function $f : \{0,1\}^n \rightarrow$

$\{0,1\}$ with the formula $f(p_1, ..., p_n)$, which we have defined to have length $n+1$, whereas some Boolean functions require exponentially-long propositional formulas [66], and thus a formula equivalent to $Bool(f(p_1, ..., p_n), \{1\})$ may be exponentially long.

The above, 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 $f(p_1, ..., p_n)$ 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. So 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 (see e.g., Chapter 6 in [9]), and thus we can choose the function f above to be PSPACE-complete, and 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 2.3 *There exists a set of functions $\mathcal{F}$ and $\mathrm{LTL}[\mathcal{F}]$ 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 $|\varphi_n|$.*

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

The *automata-theoretic* approach uses the theory of automata as a unifying paradigm for system specification, verification, and synthesis [69, 72]. 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

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

and containment. Automata-based methods have been implemented in both academic and industrial automated-verification tools (c.f., FormalCheck and SPIN) [53, 41]. In this section we describe an automata-theoretic framework for reasoning about $\text{LTL}[\mathcal{F}]$ specifications.

One approach is to develop a framework that is based on *weighted automata*. Like $\text{LTL}[\mathcal{F}]$ formulas, weighted automata map words to values that are richer than `True` and `False`, and in particular can map words to values in $[0, 1]$ [61, 28]. 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 [45]. Moreover, the semantics of weighted automata is typically simple (e.g., consists of operations on a semi-ring), 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., [44, 36]), are not suitable. Accordingly, a second approach, which is the one we follow, is to try and reduce the quantitative questions about $\text{LTL}[\mathcal{F}]$ to questions about Boolean automata. The fact that $\text{LTL}[\mathcal{F}]$ 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 2.2 would result in algorithms that are exponentially more complex than these 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 [71]. 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) *nondeterministic Büchi automaton* (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 $i \geqslant 0$, 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 [71], each state of the automaton is associated with a set of formulas, and the NGBA accepts a computation from a state q iff 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 assures 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 $\mathrm{LTL}[\mathcal{F}]$ 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 iff $[\![\pi, \varphi]\!] \in P$.

We note that a similar approach is taken in [35], where LTL formulas are interpreted over quantitative systems. The important difference is that the values in our construction arise from the formula and the functions it involves, whereas in [35] they are induced by the values of the atomic propositions.

Finally, we remark that in case the input for our construction is in fact 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 $\mathrm{LTL}[\mathcal{F}]$ formula).

Theorem 2.4 *Let φ be an $\mathrm{LTL}[\mathcal{F}]$ 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 $[\![\pi, \varphi]\!] \in P$ iff $\mathcal{A}_{\varphi,P}$ 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 2.1, the set $V(\psi)$ is finite for every formula ψ, and 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 $\psi = \mathtt{True}$, then $g(\psi) = 1$, and if $\psi = \mathtt{False}$ 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 $g' \in \delta(g, \sigma)$ iff the following hold:

- $\sigma = \{p \in AP : g(p) = 1\}$.

- For all $\mathsf{X}\psi_1 \in cl(\varphi)$, we have $g(\mathsf{X}\psi_1) = g'(\psi_1)$.

- For all $\psi_1\mathsf{U}\psi_2 \in cl(\varphi)$, we have $g(\psi_1\mathsf{U}\psi_2) = \max\{g(\psi_2), \min\{g(\psi_1), g'(\psi_1\mathsf{U}\psi_2)\}\}$.

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

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

Since the above 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. Indeed, consider a sub-formula 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 $i \leqslant j$ such that $[\![\pi^j, \psi_1\mathsf{U}\psi_2]\!] = [\![\pi^j, \psi_2]\!]$. So 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 $\rho = g_1, g_2, ...$ be an accepting run of $\mathcal{A}_{\varphi,P}$ on π, and let $h_1, h_2, ...$ 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 $h_i(p) = 1$ iff $[\![\pi^i, p]\!] = 1$ iff $p \in \pi_i$ iff $g_i(p) = 1$, and we are done.

- Let $\psi = f(\psi_1, ..., \psi_k)$. 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 2.5 below, we prove that since ρ is an accepting run, then
$$g_i(\psi_1 \mathsf{U} \psi_2) = \max_{i \leqslant j}\{\min\{g_j(\psi_2), \min_{i \leqslant k < j}\{g_k(\psi_1)\}\}\}.$$
By the induction hypothesis, this expression equals
$$\max_{i \leqslant j}\left\{\min\left\{[\![\pi^j, \psi_2]\!], \min_{i \leqslant k < j}\left\{[\![\pi^k, \psi_1]\!]\right\}\right\}\right\},$$
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 also the set of states Q is contained in the set of functions $g : cl(\varphi) \to V^+(\varphi)$. We first claim that $|V^+(\varphi)| \leqslant 2^n$. The proof is similar to the proof of Lemma 2.1 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 $|V^+(\psi_i)| \leqslant 2^{|\psi_i|}$, implying that $|V^+(\varphi)| \leqslant 2 \cdot 2^{\sum_{i=1}^k |\psi_i|} = 2^n$, so we are done.

Now, since $|cl(\varphi)| \leqslant n$ and $|V^+(\varphi)| \leqslant 2^n$, 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 $\psi_1 \mathsf{U} \psi_2$, and hence bounded by n.

Lemma 2.5 *Using the notations in the proof of Theorem 2.4, we have that*
$$g_i(\psi_1 \mathsf{U} \psi_2) = \max_{i \leqslant j}\left\{\min\left\{g_j(\psi_2), \min_{i \leqslant k < j}\left\{g_k(\psi_1)\right\}\right\}\right\}.$$

Proof: Since ρ is a legal run, then for every $i \in \mathbb{N}$, it holds that
$$g_i(\psi_1 \mathsf{U} \psi_2) = \max\left\{g_i(\psi_2), \min\left\{g_i(\psi_1), g_{i+1}(\psi_1 \mathsf{U} \psi_2)\right\}\right\}. \quad (*)$$

We prove the lemma by proving two inequalities. We start by proving that
$$g_i(\psi_1 \mathsf{U} \psi_2) \leqslant \max_{i \leqslant j}\left\{\min\left\{g_j(\psi_2), \min_{i \leqslant k < j}\left\{g_k(\psi_1)\right\}\right\}\right\}.$$

Let g_i be a state in ρ. Since ρ is accepting, there exists an index l such that $i \leqslant l$ 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 $g_i(\psi_1 \mathsf{U} \psi_2) = \max\left\{g_i(\psi_2), \min\left\{g_i(\psi_1), g_{i+1}(\psi_1 \mathsf{U} \psi_2)\right\}\right\}$. Plugging the induction hypothesis for g_{i+1}, we get that

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

It is a simple exercise to verify that the latter equals

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

and we are done.

For the second direction, we proceed similarly. Let l be the minimal index $i \leqslant l$ 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

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

Assume by way of contradiction that this does not hold. Thus, there exists some $k > i$ such that $g_i(\psi_2) = g_i(\psi_1 \mathsf{U} \psi_2) < \min\left\{g_k(\psi_2), \min_{i \leqslant j<k}\left\{g_j(\psi_1)\right\}\right\}$. From $(*)$ we make two observations. First, for every state g from which there exists an outgoing edge, we have that $g(\psi_1 \mathsf{U} \psi_2) \geqslant g(\psi_2)$ (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 $g(\psi_1 \mathsf{U} \psi_2) \geqslant \max\left\{g(\psi_1), g(\psi_2)\right\}$ (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 $g_{i+1}(\psi_1 \mathsf{U} \psi_2) \leqslant g_i(\psi_1 \mathsf{U} \psi_2)$. Applying the same logic inductively, using the assumption that $\min\left\{g_i(\psi_1), ..., g_{k-1}(\psi_1), g_k(\psi_2)\right\}$ is greater than $g_i(\psi_1 \mathsf{U} \psi_2)$, we get that $g_i(\psi_1 \mathsf{U} \psi_2) \geqslant g_k(\psi_1 \mathsf{U} \psi_2) \geqslant g_k(\psi_2) > g_i(\psi_1 \mathsf{U} \psi_2)$, 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 $\leqslant$ instead of $\geqslant$ in the induction assumption. □ □

Remark 2.6 Observe that while the size of $\mathcal{A}_{\varphi,P}$ described in Theorem 2.4 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 become the computational bottleneck of the computation, as was the case in Section 2.3.1.

Remark 2.7 The construction described in the proof of Theorem 2.4 is such that selecting the set of initial states allows us to specify any (propositional) condition regarding the sub-formulas of φ. A simple extension of this idea allows us to consider a set of formulas $\{\varphi_1, ..., \varphi_m\} = \Phi$ and a predicate $P \subseteq [0,1]^m$, and to construct an NGBA that accepts a computation π iff $\langle [\![\pi, \varphi_1]\!], ..., [\![\pi, \varphi_n]\!] \rangle \in P$. Indeed, the state space of the product consists of functions that map all the formulas in Φ to their satisfaction values, and we only have to choose as the initial states these functions g for which $\langle g(\varphi_1), ..., g(\varphi_n) \rangle \in P$. As we shall see in Section 2.5, this allows us to use the automata-theoretic approach also 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 2.4 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 $[\![\pi, \varphi]\!] \geqslant v$. 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 $\forall\pi, [\![\pi,\varphi]\!] \geqslant v$ iff $\neg(\exists\pi,\ [\![\pi,\varphi]\!] < v)$. 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, $\max\{[\![\pi,\varphi_1]\!] - [\![\pi,\varphi_2]\!] : \pi \text{ is a computation}\} = 2\cdot\max\left\{[\![\pi,\varphi_1 \oplus_{\frac{1}{2}} \neg\varphi_2]\!] : \pi \text{ is a computation}\right\} - 1$. 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 2.7 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 2.4 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 case 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 multi-valued monitoring [27], 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 [65]. 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 [51] and its extension to NGBAs [48]. 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 $[\![\pi,\varphi]\!] \geqslant v$. 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 2.8 Recall that a solution to search and optimization problems that correspond to the above decision problems involves the solution of multiple decision problems. Observe that for all the problems described above, the solution to the multiple decision problems involve reasoing 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, so 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 3.1–3.3 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, where 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 non-weighted systems, except that for an atomic proposition p, we have $[\![\pi, p]\!] = L(\pi_0)(p)$.

It is not hard to extend the construction of $\mathcal{A}_{\varphi,P}$, as described in the proof of Theorem 2.4, 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 $c \in \Upsilon$ iff 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 where the values for the atomic propositions are known, and in particular model checking, the solutions to the search questions are similar to the ones described for $\text{LTL}[\mathcal{F}]$ with Boolean atomic propositions. Formally, we have the following theorem.

Theorem 3.1 *Let $\mathcal{K}$ be a weighted Kripke structure, φ be an $\text{LTL}[\mathcal{F}]$ 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 $[\![\pi,\varphi]\!] \in P$ iff $\mathcal{A}_{\mathcal{K},\varphi,P}$ 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 $\text{LTL}[\mathcal{F}]$ over Finite Computations

In the Boolean setting, one can interpret LTL over finite computations [58]. The main change in the semantics is the evaluation of the X operator in the last position in the computation, where $\mathsf{X}\varphi$ can be arbitrarily defined as `True` or as `False`.

Similarly, one can interpret $\text{LTL}[\mathcal{F}]$ 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 1, with two exceptions. First, $[\![\pi, \varphi_1 \mathsf{U} \varphi_2]\!] = \max_{0 \leqslant i \leqslant n} \{\min\{[\![\pi^i, \varphi_2]\!], \min_{0 \leqslant j < i} [\![\pi^j, \varphi_1]\!]\}\}$. Second, $[\![\pi, \mathsf{X}\varphi]\!]$ is $[\![\pi^1, \varphi]\!]$ when $n \geqslant 1$ (that is, when the computation is of length at least 2), and is 0 when $n = 0$ (that is, 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 $\text{LTL}[\mathcal{F}]$ over finite computations via the same approach we took in Section 2.4, by translating an $\text{LTL}[\mathcal{F}]$ formula to an automaton. The construction is similar to the one presented in Theorem 2.4, with the difference being that we construct an NFA, rather than an NGBA. Specifically, given an $\text{LTL}[\mathcal{F}]$ 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 2.4. 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 $g(\psi_2) = g(\psi_1 \mathsf{U} \psi_2)$

- For all $\mathsf{X}\psi \in cl(\varphi)$ we have $g(\mathsf{X}\psi) = 0$.

The correctness of the construction can be proved similarly to Theorem 2.4.

3.3 Formalizing Quality with Branching Temporal Logics

Formulas of $\mathrm{LTL}[\mathcal{F}]$ specify on-going behaviors of linear computations. A Kripke structure is not linear, and the way we interpret $\mathrm{LTL}[\mathcal{F}]$ 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 [32].

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

In [33], the authors 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 naive attempt to use this technique in order to model check $\mathrm{CTL}^\star[\mathcal{F}]$ fails, as the satisfaction value of $\mathrm{LTL}[\mathcal{F}]$ formulas is not Boolean, and hence they cannot be replaced by atomic propositions. Fortunately, having solved the $\mathrm{LTL}[\mathcal{F}]$ 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, where the value of a fresh weighted atomic proposition is found by solving the search variant of the $\mathrm{LTL}[\mathcal{F}]$ model-checking problem. Hence, the model-checking problem for $\mathrm{CTL}^\star[\mathcal{F}]$ is PSPACE-complete, as are the ones for $\mathrm{LTL}[\mathcal{F}]$ and for CTL*. A direction for future research is to study fragments of $\mathrm{CTL}^\star[\mathcal{F}]$ for which model checking is simpler. In particular, our initial results show that for $\mathrm{CTL}[\mathcal{F}]$, 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 [34]. Thus, the model-checking problem for $\mathrm{CTL}[\mathcal{F}]$ is in P.

More challenging is the handling of the other search problems. There, the solution involves a translation of $\mathrm{CTL}^\star[\mathcal{F}]$ formulas to tree automata. Since the automata-theoretic approach for CTL* satisfiability and synthesis has the Vardi-Wolper construction at its heart [52], this is possible. Below we elaborate on the solution.

Consider a $\mathrm{CTL}^\star[\mathcal{F}]$ 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 iff $[\![\tau, \varphi]\!] \geqslant v$. The construction is similar to that in [52]. 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 $\mathrm{CTL}^\star[\mathcal{F}]$, we only need to observe that the inner NBAs can be obtained with the construction in Theorem 2.4. 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 $\mathrm{CTL}^\star$ synthesis in the Boolean case, for which a matching 2-EXPTIME lower bound is known[50]. We conclude with the following theorem.

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

3.4 The Fragment LTL$^\triangledown$

We define below a fragment of $\mathrm{LTL}[\mathcal{F}]$ that covers the possible linear approaches of formalizing quality as a value between true and false. Formally, we define the logic $\mathrm{LTL}^\triangledown$ as a special case of $\mathrm{LTL}[\mathcal{F}]$, where the set of functions $\mathcal{F}$ contains the standard Boolean operators and the unary operators $\triangledown_\lambda$, $\blacktriangledown_\lambda$, and $⧩_\lambda$, defined as follows. Let $x \in [0,1]$. Then,

- $\triangledown_\lambda(x) = \lambda \cdot x$
- $\blacktriangledown_\lambda(x) = \lambda x + (1-\lambda)$
- $⧩_\lambda(x) = \lambda \cdot x + (1-\lambda)/2$

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

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 $\llbracket \cdot,\cdot \rrbracket_{[-1,0]}$, whereby $\llbracket \pi, \texttt{True} \rrbracket_{[-1,0]} = 0$ and $\llbracket \pi, \texttt{False} \rrbracket_{[-1,0]} = -1$. With this view, we have the operator $\blacktriangledown_\lambda$ with the semantics $\llbracket \pi, \blacktriangledown_\lambda\varphi \rrbracket_{[-1,0]} = \lambda\llbracket \pi, \varphi \rrbracket_{[-1,0]}$. 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 $\llbracket \pi, \neg\varphi \rrbracket_{[-1,0]} = -(1 + \llbracket \pi, \varphi \rrbracket_{[-1,0]})$. It is easy to check that in the framework of $\mathrm{LTL}[\mathcal{F}]$, whereby $\llbracket \pi, \texttt{True} \rrbracket = 1$ and $\llbracket \pi, \texttt{False} \rrbracket = 0$, 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 $\llbracket \cdot,\cdot \rrbracket_{[-1,1]}$ whereby $\llbracket \pi, \texttt{True} \rrbracket_{[-1,1]} = 1$ and $\llbracket \pi, \texttt{False} \rrbracket_{[-1,1]} = -1$. This captures the intuition that `False` is the opposite of `True` (rather than the lack of `True`). In this semantics, $\llbracket \pi, \neg\varphi \rrbracket_{[-1,1]} = -\llbracket \pi, \varphi \rrbracket_{[-1,1]}$

and the operator $[\![\pi, \boldsymbol{\nabla}_\lambda\varphi]\!]_{[-1,1]} = \lambda[\![\pi,\varphi]\!]$. We use $\boldsymbol{\nabla}$ 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 $\boldsymbol{\nabla}_\lambda(x) = \lambda \cdot x + (1-\lambda)/2$.

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 2.2, is polynomial. Indeed, for such predicates we have that $\mathit{Bool}(\varphi_1 \mathsf{U} \varphi_2, [c,1]) = \mathit{Bool}(\varphi_1, [c,1])\mathsf{U}\mathit{Bool}(\varphi_2,[c,1])$, and $\mathit{Bool}(\triangledown_\lambda\varphi, [c,1]) = \mathit{Bool}(\varphi, [\frac{c}{\lambda}, 1])$ if $\frac{c}{\lambda} \leqslant 1$ and `False` otherwise. The translation of $\blacktriangledown$ and $\boldsymbol{\nabla}$ 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 we have in them. We demonstrate the usefulness of this fragment in the following examples.

Example 3.1 *Consider the specification* $\mathsf{G}(req \to grant \vee \mathsf{X}grant)$*, and suppose 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*

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

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 above formula to

$$\mathsf{G}(req \to ((grant \wedge \neg\mathsf{X}grant) \vee \boldsymbol{\nabla}_{\frac{3}{4}}(\neg grant \wedge \mathsf{X}grant) \vee \boldsymbol{\nabla}_{\frac{1}{2}}(grant \wedge \mathsf{X}grant))).$$

Note that here, $(grant \wedge \mathsf{X}grant)$ *gives satisfaction value of* $\frac{3}{4}$ *(according to the semantics of* $\boldsymbol{\nabla}$ *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 $\mathrm{LTL}^{\triangledown}$ can be used for vacuity testing and coverage, as well as prioritizing safety requirements.

Example 3.2 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 [46]. For example, after a system is proven to satisfy the specification $\mathsf{G}(req \to \mathsf{F}grant)$, vacuity-detection algorithms would mutate the specification to $\mathsf{G}(req \to \mathtt{False})$ or to $\mathsf{G}(\mathtt{True} \to \mathsf{F}grant)$ 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 example above, it is more alarming to discover that the specification has been satisfied in a system where no requests are issued, than to discover that the specification has been satisfied in a system in which infinitely many grants are issued, Using $\mathrm{LTL}^{\triangledown}$, 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 $\mathsf{G}(req \to \mathsf{F}grant)$ to $\mathsf{G}((\triangledown_{\frac{1}{3}} \neg req) \vee \mathsf{F}grant)$, which would get satisfaction value $\frac{1}{3}$ in computations in which it is satisfied only thanks to the fact there are only finitely many requests.

Example 3.3 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"? Indeed, 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 have 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 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 then 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 $\mathbb{N} = \{0, 1, ...\}$. 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:

- `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 $\text{LTL}[\mathcal{F}]$ 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 follows, where $\pi^i = \pi_i, \pi_{i+1}, \ldots$

Formula	Satisfaction value
$\llbracket \pi, \texttt{True} \rrbracket$	1
$\llbracket \pi, p \rrbracket$	$1 \quad \text{if } p \in \pi_0$ $0 \quad \text{if } p \notin \pi_0$
$\llbracket \pi, \neg\varphi_1 \rrbracket$	$1 - \llbracket \pi, \varphi_1 \rrbracket$
$\llbracket \pi, \varphi_1 \vee \varphi_2 \rrbracket$	$\max\{\llbracket \pi, \varphi_1 \rrbracket, \llbracket \pi, \varphi_2 \rrbracket\}$
$\llbracket \pi, \mathsf{X}\varphi_1 \rrbracket$	$\llbracket \pi^1, \varphi_1 \rrbracket$
$\llbracket \pi, \varphi_1 \mathsf{U} \varphi_2 \rrbracket$	$\sup\limits_{i \geqslant 0}\{\min\{\llbracket \pi^i, \varphi_2 \rrbracket, \min\limits_{0 \leqslant j < i}\{\llbracket \pi^j, \varphi_1 \rrbracket\}\}\}$
$\llbracket \pi, \varphi_1 \mathsf{U}_\eta \varphi_2 \rrbracket$	$\sup\limits_{i \geqslant 0}\{\min\{\eta(i)\llbracket \pi^i, \varphi_2 \rrbracket, \min\limits_{0 \leqslant j < i}\{\eta(j)\llbracket \pi^j, \varphi_1 \rrbracket\}\}\}$

Table 2: The semantics of $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$.

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 i-th position.

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

Remark 4.1 A more restricted way of future-discounting can be captured with a *discounted* X *operator* X_λ where $\llbracket \pi, \mathsf{X}_\lambda\varphi \rrbracket = \lambda\llbracket \pi^1, \varphi \rrbracket$. 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. Indeed, for the semantics of $\varphi \mathsf{U} \psi$ this is trivial. For the semantics of $\varphi \mathsf{U}_\eta \psi$, recall that $\llbracket \pi, \varphi_1 \mathsf{U}_\eta \varphi_2 \rrbracket = \sup\limits_{i \geqslant 0} H$, where H is the term

[7] Observe that in our semantics the satisfaction value of future events tends to 0. One may think of scenarios where 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.

$\min\{\eta(i)[\![\pi^i,\varphi_2]\!], \min_{0\leqslant j<i}\{\eta(j)[\![\pi^j,\varphi_1]\!]\}\}$. Since η is decreasing, it follows that for a large enough i_0, H is decreasing as a function of $i \geqslant i_0$. Thus, the $\sup$ is attained within a finite prefix, and is therefore 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 $[\![\mathcal{K},\varphi]\!] = \inf\left\{[\![\pi,\varphi]\!] : \pi \text{ is a computation of } \mathcal{K}\right\}$.

Example 4.1 *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, but the longer it runs, 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 (that is, a measure of the amount of correct bits) can be specified by the formula $\varphi = \mathsf{GF}_\eta(init \wedge \neg\mathsf{F}_\mu terminate)$ 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 (that is, not terminated) in between these initiations. Note that the "worst case" nature of $\mathrm{LTL}^{disc}[\mathcal{D}]$ 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 $L(\mathcal{A}_{\varphi,v}) \neq \emptyset$ iff there exists a computation π such that $[\![\pi,\varphi]\!] > v$. The challenge here is that φ has infinitely many satisfaction values, naively 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 [60]. 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 $\theta \in \mathcal{B}^+(X)$ iff the truth assignment that assigns *true* to the members of Y and assigns *false* to the members of $X \backslash Y$ 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.
- $E \subseteq \bigcup_{l \geqslant 0} (Q_l \times \{l\}) \times (Q_{l+1} \times \{l+1\})$ is such that for every state $q \in Q_l$, the set $\left\{q' \in Q_{l+1} : (\langle q, l \rangle, \langle q', l+1 \rangle) \in E\right\}$ 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, where 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 $1 \leqslant i \leqslant k$ 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 $\leqslant$ over the sets, and for every $1 \leqslant i, j \leqslant k$, if $q \in Q_i$, $s \in Q_j$, and $s \in \delta(q, \sigma)$ for some $\sigma \in \Sigma$, then $Q_j \leqslant Q_i$. 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 iff this set is accepting.

4.2.2 From $\text{LTL}^{\text{disc}}[\mathcal{D}]$ 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 $\psi > t$ iff $[\![\pi, \psi]\!] > t$. 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 non-empty iff there is a lasso witness for its non-emptiness.

Defining the appropriate transition function for the AWA follows the semantics of $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$ in the expected manner. A naive 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 4.5, 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 4.2 *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 "iff" criterion.

- If φ is of the form $\mathtt{True}, \mathtt{False}$, 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 $\varphi^{<1} = \psi_1^{<1} \wedge \psi_2^{<1}$. Indeed, for every computation π we have that $[\![\pi,\varphi]\!] > 0$ iff either $[\![\pi,\psi_1]\!] > 0$ or $[\![\pi,\psi_2]\!] > 0$, and $[\![\pi,\varphi]\!] < 1$ iff 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 $[\![\pi,\varphi]\!] > 0$ iff there exists $i \geqslant 0$ such that $[\![\pi^{i},\psi_2]\!] > 0$ and for every $0 \leqslant j < i$ it holds that $[\![\pi^{j},\psi_1]\!] > 0$. This happens iff 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 $i \geqslant 0$ it holds that either $[\![\pi^i, \psi_2]\!] < 1$ or $[\![\pi^j, \psi_1]\!] < 1$ for some $0 \leqslant j < i$. Thus, for every $i \geqslant 0$, either $\pi^i \models \psi_2^{<1}$, or $\pi^j \models \psi_1^{<1}$ for some $0 \leqslant j < i$. So for every $i \geqslant 0$, either $\pi^i \not\models \neg(\psi_2^{<1})$, or $\pi^j \not\models \neg(\psi_1^{<1})$ for some $0 \leqslant j < i$. 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 $\pi \not\models (\neg(\psi_1^{<1}))\mathsf{U}(\neg(\psi_2^{<1}))$, so for every $i \geqslant 0$, either $\pi^i \models \psi_2^{<1}$ or $\pi^j \models \psi_1^{<1}$ for some $0 \leqslant j < i$. By the induction hypothesis, for every $i \geqslant 0$, either $[\![\pi^i, \psi_2]\!] < 1$ or $[\![\pi^j, \psi_1]\!] < 1$ for some $0 \leqslant j < i$. 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 (namely $\pi^0, ..., \pi^{|u|+|v|-1}$). 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 $\varphi^+ = \psi_1^+\mathsf{U}\psi_2^+$. Indeed, since $\eta(i) > 0$ for all $i \geqslant 0$, then $\varphi^+ = \psi_1\mathsf{U}\psi_2{}^+$.

  Now, $\varphi^{<1}$ is defined as follows. First, if $\eta(0) < 1$, then $\varphi^{<1} = \mathtt{True}$. If $\eta(0) = 1$, then $\varphi^{<1} = \psi_2{}^{<1}$. Indeed, 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 iff $\eta(0) = 1$ and $\pi \not\models \psi_2^{<1}$ (where 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 4.2.

Consider an $\text{LTL}^{\text{disc}}[\mathcal{D}]$ formula φ. By Lemma 4.2, 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 [70] we know that φ^+ is satisfiable iff there exists a lasso computation π that satisfies it, in which case $[\![\pi, \varphi]\!] > 0$. We thus get the following.

Corollary 4.3 *Consider an* $\text{LTL}^{\textit{disc}}[\mathcal{D}]$ *formula* φ. *There exists a computation* π *such that* $[\![\pi, \varphi]\!] > 0$ *iff there exists a lasso computation* π' *such that* $[\![\pi', \varphi]\!] > 0$, *in which case* $\pi' \models \varphi^+$ *as well.*

Remark 4.4 The curious reader may wonder why we do not prove that $[\![\pi, \varphi]\!] > 0$ iff $\pi \models \varphi^+$ for every computation π. As it turns out, a translation that is valid also for computations that are not lasso-shaped (that is, ones with no period) is not always possible. For example, as is the case with the prompt-eventuality operator of [49], 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 $f^{+k} : \mathbb{N} \to [0,1]$, where for every $i \in \mathbb{N}$, we have that $f^{+k}(i) = f(i+k)$.

Let φ be an 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 LTL$^{\text{disc}}[\mathcal{D}]$ formulas (from Classes 1 and 3) and LTL formulas (from Class 2).

Theorem 4.5 *Given an* LTL$^{\textit{disc}}[\mathcal{D}]$ *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 below. 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 [70]. 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 $p \notin \sigma$.

- $\delta(\psi_1 \vee \psi_2, \sigma) = \delta(\psi_1, \sigma) \vee \delta(\psi_2, \sigma)$.

- $\delta(\neg\psi, \sigma) = \widetilde{\delta(\psi, \sigma)}$.

- $\delta(\mathsf{X}\psi, \sigma) = \psi$.

- $\delta(\psi_1 \mathsf{U} \psi_2, \sigma) = \delta(\psi_2, \sigma) \vee (\delta(\psi_1, \sigma) \wedge (\psi_1 \mathsf{U} \psi_2))$.

Finally, the accepting Type-2 states are those that correspond to formulas of the form $\neg(\psi_1 \mathsf{U} \psi_2)$. By [70], a computation π is accepted from state $q \in Q_2$ iff $\pi \models q$.

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 $\sigma \in 2^{AP}$.

- $\delta((\texttt{True} > t), \sigma) = \left[\begin{array}{ll} \texttt{True} & \text{if } t < 1, \\ \texttt{False} & \text{if } t = 1. \end{array}\right.$

- $\delta((\texttt{False} > t), \sigma) = \texttt{False}$.

- $\delta((\texttt{True} < t), \sigma) = \texttt{False}$.

- $\delta((\texttt{False} < t), \sigma) = \left[\begin{array}{ll} \texttt{True} & \text{if } t > 0, \\ \texttt{False} & \text{if } t = 0. \end{array}\right.$

- $\delta((p > t), \sigma) = \left[\begin{array}{ll} \texttt{True} & \text{if } p \in \sigma \text{ and } t < 1, \\ \texttt{False} & \text{otherwise.} \end{array}\right.$

- $\delta((p < t), \sigma) = \left[\begin{array}{ll} \texttt{False} & \text{if } p \in \sigma \text{ or } t = 0, \\ \texttt{True} & \text{otherwise.} \end{array}\right.$

- $\delta((\psi_1 \vee \psi_2 > t), \sigma) = \delta((\psi_1 > t), \sigma) \vee \delta((\psi_2 > t), \sigma)$.

- $\delta((\psi_1 \vee \psi_2 < t), \sigma) = \delta((\psi_1 < t), \sigma) \wedge \delta((\psi_2 < t), \sigma)$.

- $\delta((\neg\psi_1 > t), \sigma) = \delta((\psi_1 < 1 - t), \sigma)$.

- $\delta((\neg\psi_1 < t), \sigma) = \delta((\psi_1 > 1 - t), \sigma)$.
- $\delta((\mathsf{X}\psi_1 > t), \sigma) = (\psi_1 > t)$.
- $\delta((\mathsf{X}\psi_1 < t), \sigma) = (\psi_1 < t)$.
- $\delta((\psi_1 \mathsf{U} \psi_2 > t), \sigma) = \left[\begin{array}{ll} \delta((\psi_2 > t), \sigma) \vee [\delta((\psi_1 > t), \sigma) \wedge (\psi_1 \mathsf{U} \psi_2 > t)] & \text{if } 0 < t < 1, \\ \texttt{False} & \text{if } t = 1, \\ \delta(((\psi_1 \mathsf{U} \psi_2)^+), \sigma) & \text{if } t = 0. \end{array}\right.$
- $\delta((\psi_1 \mathsf{U} \psi_2 < t), \sigma) = \left[\begin{array}{ll} \delta((\psi_2 < t), \sigma) \wedge [\delta((\psi_1 < t), \sigma) \vee (\psi_1 \mathsf{U} \psi_2 < t)] & \text{if } 0 < t \leqslant 1, \\ \texttt{False} & \text{if } t = 0. \end{array}\right.$
- $\delta((\psi_1 \mathsf{U}_\eta \psi_2 > t), \sigma) =$
  $\left[\begin{array}{ll} \delta((\psi_2 > \frac{t}{\eta(0)}), \sigma) \vee [\delta((\psi_1 > \frac{t}{\eta(0)}), \sigma) \wedge (\psi_1 \mathsf{U}_{\eta^{+1}} \psi_2 > t)] & \text{if } 0 < \frac{t}{\eta(0)} < 1, \\ \texttt{False} & \text{if } \frac{t}{\eta(0)} \geqslant 1, \\ \delta(((\psi_1 \mathsf{U}_\eta \psi_2)^+), \sigma) & \text{if } \frac{t}{\eta(0)} = 0 \text{ (i.e., } t = 0). \end{array}\right.$
- $\delta((\psi_1 \mathsf{U}_\eta \psi_2 < t), \sigma) =$
  $\left[\begin{array}{ll} \delta((\psi_2 < \frac{t}{\eta(0)}), \sigma) \wedge [\delta((\psi_1 < \frac{t}{\eta(0)}), \sigma) \vee (\psi_1 \mathsf{U}_{\eta^{+1}} \psi_2 < t)] & \text{if } 0 < \frac{t}{\eta(0)} \leqslant 1, \\ \texttt{True} & \text{if } \frac{t}{\eta(0)} > 1, \\ \texttt{False} & \text{if } \frac{t}{\eta(0)} = 0 \text{ (i.e., } t = 0). \end{array}\right.$

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 $\eta(0) \leqslant t$ (equivalently, $\frac{t}{\eta(0)} \geqslant 1$), then it cannot hold that $\psi_1 \mathsf{U}_\eta \psi_2 > t$, so 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 4.3, this is equivalent to the satisfiability of the former. So 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 above 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 sub formulas 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 either ψ_s is a strict subformula of ψ_q, or 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's 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 $xcl(\varphi)$, and thus there are only finitely many Type-2 states. This follows immediately from Lemma 4.2 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 $O(|\varphi|)$. Indeed, 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. Indeed, for every discounting function $\eta \in \mathcal{D}$ (that appears in φ), let $i_{t,\eta} = \max\left\{i : \frac{t}{\eta(i)} \leqslant 1\right\}$. 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 claim above. 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 $i \leqslant i_{v,\eta}$. 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. So 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 4.2 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 iff it accepts a lasso computation. Combining this observation with Theorem 4.5, we conclude with the following.

Corollary 4.6 *For an* LTL$^{disc}[\mathcal{D}]$ *formula* φ *and a threshold* $v \in [0, 1]$*, it holds that* $L(\mathcal{A}_{\varphi,v}) \neq \emptyset$ *iff 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 nondeterministic Büchi automaton (NBA, for short), yet the state blowup might be exponential [60, 15]. By carefully analyzing the AWA $\mathcal{A}_{\varphi,v}$ generated in Theorem 4.5, 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 4.7 *For an* LTL$^{disc}[\mathcal{D}]$ *formula* φ *and* $v \in [0, 1]$*, the* AWA $\mathcal{A}_{\varphi,v}$ *constructed in Theorem 4.5 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 [38], the AWA is translated to an NGBA (nondeterministic generalized Büchi autoamton, 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 4.5, it holds that a path π is accepted from both states, iff 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 $|\mathcal{A}|$, so the size of the obtained NBA is also $|\mathcal{A}|^{O(|\varphi|)}$. □

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 $[\![\mathcal{K}, \varphi]\!] \geqslant v$.

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

In the case of $\mathrm{LTL}[\mathcal{F}]$, where every formula only has exponentially many satisfaction values, solutions to the decision problems imply solutions to the search problems. On the other hand, an $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$ formula might have infinitely many satisfaction values. Hence, one cannot reduce an $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$ search problem to a finite set of decision problems with respect to specific thresholds. Below we solve the decision problems for $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$, leaving the search problems open. Moreover, note that in our construction in Theorem 4.5, we only consider strict inequalities. Therefore, our solutions to the threshold problems are often only for strict inequalities, or, in dual problems, only for non-strict 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 non-strict inequality.

The difficulty in handling non-strict thresholds stems from the lack of a lasso-witness (see Remark 4.4). We note that similar problems are known to be notoriously difficult, and are encountered in other settings: in multi-objective MDPs [19], the need for infinite-memory strategies renders certain problems open, and in [14], the discounted-sum problem remains open when considering words that are not eventually periodic.

- **Satisfiability and validity.** The NBA obtained in Lemma 4.7 can be directly used for solving the strict threshold-satisfiability and strict threshold-validity problems: given an $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$ 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 $[\![\pi, \varphi]\!] \geqslant v$ for every computation $\pi \in (2^{AP})^{\omega}$.

  Note that solving the non-strict version of the problems remains an open problem.

- **Implication and equivalence.** In Section 2.5, we solve the implication problem for $\mathrm{LTL}[\mathcal{F}]$ by combining the given formulas using the average operator. In the context of $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$, 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 [72]. Consider a Kripke structure $\mathcal{K}$, an $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$ 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, $L(\mathcal{A}_{\neg\varphi,1-v}) \cap L(\mathcal{K}) \neq \emptyset$ iff there exists a lasso computation π that is induced by $\mathcal{K}$ such that $[\![\pi, \varphi]\!] < v$, which happens iff it is not true that $[\![\mathcal{K}, \varphi]\!] \geqslant v$.

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 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 below 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 $v \geqslant 0$, 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 4.8 *Consider an* $\text{LTL}^{\mathit{disc}}[\mathcal{D}]$ *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* $[\![\tau, \varphi]\!] \geqslant v$*.*

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 $[\![\pi, \varphi]\!] \geqslant v$ (note the non-strict 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 [65]), we can generate from $\mathcal{A}_{\varphi,v}$ a deterministic tree automaton $\mathcal{D}$ that accepts a 2^O-labeled 2^I-tree iff all the paths along the tree are accepted in $\mathcal{A}_{\varphi,v}$. Moreover, it is shown in [39] 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 non-empty, and a witness to its non-emptiness is a regular tree that induced by a transducer all whose computations satisfy $[\![\pi, \varphi]\!] \geqslant v$. □

We note that this solution is only partial, as there might be an I/O-transducer, all of whose computations π satisfy $[\![\pi, \varphi]\!] \geqslant v$, 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 non-strict inequality.

Remark 4.9 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 $v_2 - v_1 \leqslant \epsilon$, 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 $[v_1 - \epsilon, v_1 + \epsilon]$. In [63], the authors 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. The complexities of solving the decision problems for $\text{LTL}^{\text{disc}}[\mathcal{D}]$ depend, thus, 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) [67, 24]. 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 $\text{LTL}^{\text{disc}}[E]$.

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

For an $\text{LTL}^{\text{disc}}[E]$ formula φ, let $F(\varphi)$ be the set $\{\lambda : \text{the operator } \mathsf{U}_{\exp_\lambda} \text{ appears in } \varphi\}$. 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 $\text{LTL}^{\text{disc}}[\mathcal{D}]$ formula φ and a threshold v is the generation of an AWA $\mathcal{A}_{\varphi,v}$ (Section 4.2). We will show below that for $\text{LTL}^{\text{disc}}[E]$ 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 4.5 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 4.10 *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

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

- $$\delta((\psi_1 \mathsf{U}_{\exp_\lambda} \psi_2 < t), \sigma) = \left[\begin{array}{ll} \delta((\psi_2 < t), \sigma) \wedge & \\ \quad [\delta((\psi_1 < t), \sigma) \vee (\psi_1 \mathsf{U}_{\exp_\lambda} \psi_2 < \frac{t}{\lambda})] & \text{if } 0 < t \leqslant 1, \\ \texttt{False} & \text{if } t = 0\ . \end{array}\right.$$

The correctness and finiteness of the construction follows from Theorem 4.5, with the observation above. 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 $O(|\varphi|)$. Indeed, 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, so 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 case $\psi = \psi_1 \mathsf{U}_{\exp_\lambda} \psi_2$), 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 case $\psi = \neg\psi_1$, or to a "fresh" threshold, in case 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.

W.l.o.g. assume that all the discounting factors in $F\varphi$ have the same denominator $\widehat{q}$. Indeed, 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 above, 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 $i \leqslant \log_\lambda(t_0)$. 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 $p_0 \geqslant 1$) and $\lambda = \frac{\widehat{q}-1}{\widehat{q}}$ (recall that all factors in $F\varphi$ have denominator $\widehat{q}$, and 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 $x \geqslant 0$, which can be proved by simple analysis.

Next, we consider the threshold of the states that can be reached as described above. These thresholds are of the form $\frac{t_0}{\lambda^j} = \frac{p_0\widehat{q}^j}{q_0 p^j}$ for $j \leqslant \widehat{q}\log q_0$, where $\lambda = \frac{p}{\widehat{q}}$. Since $p < \widehat{q}$, we conclude that the denominator of these threshold 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 where 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, and thus the analysis above can be repeated. Recall that the number of changes to a deeper formula is $n = O(|\varphi|)$. Accordingly, the sequence $a_1, ..., a_n$ 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 $1 \leqslant i < n$. It is easy to see that this sequence is increasing, and that $a_i = q_0^{\left((\widehat{q}\cdot\log\widehat{q}+1)^i\right)}$.

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

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

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 4.10 is singly-exponential in the description of φ and v, the values of the thresholds that appear in the states may be doubly-exponential. Indeed, our bound on the maximal denominator 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 [1], we are able to compare thresholds to 1 in PSPACE, and 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 nodes on its inputs.

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

Lemma 4.11 *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 4.12 *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 $b_r b_{r-1} ... b_0$, 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 $0 \leqslant m \leqslant r$ (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 $0 \leqslant m \leqslant r$, and consider the binary expansion $d_t d_{t-1} ... d_0$ 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 4.11 a circuit for $2^{(2^s)}$ of size $s+1$, by first constructing the constant $l = 2$ (in the notations of Lemma 4.11) 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 4.13 *There exists a representation of $\mathcal{A}_{\varphi,v}$, the* AWA *constructed as per the proof of Lemma 4.10, 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 4.10, 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 $i \leqslant \widehat{q}(\widehat{q}\log\widehat{q}+1)^n \log q_0$. Indeed, 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 above is singly-exponential in $|\langle\varphi\rangle|$ and $|\langle v\rangle|$, so 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 above, 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 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 $h(v) \geqslant 1$. 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 [1].

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 $h(v) \geqslant 1$ iff $h_N(v) \geqslant h_D(v) > 0$, or $h_N(v) \leqslant h_D(v) < 0$. Thus, checking whether $h(v) \geqslant 1$ can be reduced to two tests: first, whether $h_D(v) > 0$, and second, whether $h_N(v) - h_D(v) \geqslant 0$ (if $h_D(v) > 0$) or whether $h_N(v) - h_D(v) \leqslant 0$ (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 4.12.
- If $h(v) = f(h'(v))$, then $h_D(v) = h'_D(v)$, so 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, and then use a

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

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 4.12. Next, let $b_r b_{r-1} ... b_0$ be the binary expansion of i, then $q^i = \prod_{j=0}^{r} q^{b_j 2^j}$. From Lemma 4.11, 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 $0 \leqslant j \leqslant r$ 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 $h_D(v) = p_0$, and we construct a circuit similarly to $h_D(v)$, using Lemma 4.12.
- If $h(v) = f(h'(v))$, then we have $h_N(v) = h'_D(v) - h'_N(v)$. 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 [1], it is shown that deciding whether the value of a circuit is nonnegative is in PSPACE. As we showed above, $h(v) \geqslant 1$ iff either $h_D(v) > 0$, and $h_N(v) - h_D(v) \geqslant 0$, or $h_D(v) < 0$ and $h_N(v) - h_D(v) \leqslant 0$. Note that all these tests can be easily reduced to deciding non-negativity of a circuit. For example, checking whether $h_N(v) - h_D(v) \leqslant 0$ amounts to constructing a circuit for $h_D(v) - h_N(v)$ using a subtraction gate, and checking non-negativity. This concludes the proof. □ □

4.4.3 Solving the decision problems for $\mathrm{LTL}^{\textbf{disc}}[E]$

Using the succinct $\mathcal{A}_{\varphi,v}$, as per Lemmas 4.10 and 4.13, we can construct an equivalent NBA, as per Lemma 4.7, and solve the satisfiability and model-checking problems in PSPACE.

Lemma 4.14 *For an* $\mathrm{LTL}^{disc}[E]$ *formula* φ *and* $v \in [0, 1]$, *the* AWA $\mathcal{A}_{\varphi,v}$ *constructed in Lemma 4.10 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 4.7.

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 4.13, 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 4.15 *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 4.16 □

Theorem 4.16 *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 4.14, 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 4.16 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 (non-discounting) accumulative values, where, when decidable, involves a doubly-exponential dependency on the size of the system [13].

5 Extending Temporal Quality

In this section we consider extensions of the temporal-quality setting. Similarly 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 $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$ 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, $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$ also naturally fits weighted systems, where 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 $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$ 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. Indeed, 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

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

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*) [56]. 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 $[\![\pi^i, \mathsf{Y}\varphi]\!] = [\![\pi^{i-1}, \varphi]\!]$ if $i > 0$, and $\texttt{False}$ otherwise, and $[\![\pi^i, \varphi\mathsf{S}\psi]\!] = \texttt{True}$ if there exists $0 \leqslant j \leqslant i$ such that $\pi^j \models \psi$ and $\pi^k \models \varphi$ for every $j < k \leqslant i$.

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

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

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 $\text{LTL}^{\text{disc}}[\mathcal{D}]$, our solution for the $\text{PLTL}^{\text{disc}}[\mathcal{D}]$ model-checking problem is by translating $\text{PLTL}^{\text{disc}}[\mathcal{D}]$ 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 $\text{PLTL}^{\text{disc}}[\mathcal{D}]$ with exponential discounting, namely $\text{PLTL}^{\text{disc}}[E]$, is in PSPACE.

As we now show, the construction we use when working with the "infinite future" U_η operator is similar to that of the one we use for the "finite past" S_η operator. The key for this somewhat surprising similarity is the fact 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 $\text{LTL}^{\text{disc}}[\mathcal{D}]$, the construction extends the construction for the Boolean case. It is not hard to extend Lemma 4.2 and generate Boolean PLTL formulas for satisfaction values in $\{0, 1\}$.

Given a $\text{PLTL}^{\text{disc}}[\mathcal{D}]$ formula φ and a threshold $t \in [0, 1)$, we construct a 2AWA as in Theorem 4.5 with the following additional transitions:[10]

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

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

[10] In addition, the atoms in all the transitions in Theorem 4.5 are adjusted to the 2AWA syntax by replacing each atom q by the atom $\langle 1, q \rangle$.

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 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.

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

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 in expense of other processes. This can be captured by the formula* $\varphi = \texttt{True}\mathsf{O}_{\eta,\frac{1}{2}}\mathrm{defrag}$. *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 4.5 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.

Indeed, 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 4.5, transitions from the state e.g., $(\psi_1\mathsf{U}_\eta\psi_2 < t)$ 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 below). 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:

- $\delta((\psi_1\mathsf{O}_{\eta,z}\psi_2 > t),\sigma) = \left[\begin{array}{ll} \delta((\psi_2 > \tau),\sigma) \vee & \\ \quad [\delta((\psi_1 > \tau),\sigma) \wedge (\psi_1\mathsf{O}_{\eta^{+1},z}\psi_2 > t)] & \text{if } 0 \leqslant \tau < 1, \\ \texttt{False} & \text{if } \tau \geqslant 1, \\ \texttt{True} & \text{if } \tau < 0. \end{array}\right.$

- $\delta((\psi_1 \mathsf{O}_{\eta,z}\psi_2 < t),\sigma) = \left[\begin{array}{ll} \delta((\psi_2 < \tau),\sigma)\wedge & \\ \;[\delta((\psi_1 < \tau),\sigma) \vee (\psi_1\mathsf{O}_{\eta^{+1},z}\psi_2 < t)] & \text{if } 0 < \tau \leqslant 1, \\ \texttt{True} & \text{if } \tau > 1, \\ \texttt{False} & \text{if } \tau \leqslant 0. \end{array}\right.$

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 $[\![\pi, \psi_1\mathsf{O}_{\eta,z}\psi_2]\!] > t$ iff π is accepted from the state $(\psi_1\mathsf{O}_{\eta,z}\psi_2 > t)$.

First, if $z = t$, then $[\![\pi, \psi_1\mathsf{O}_{\eta,z}\psi_2]\!] > t$ iff $[\![\pi, \psi_1\mathsf{U}\psi_2]\!] > t$ (this follows directly from the semantics of O). By the definition of the transitions above, 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 4.5. Next, assume that $z \neq t$.

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

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

Finally, if $0 \leqslant \tau < 1$, then similarly to the case of U_η, we have that $[\![\pi, \psi_1\mathsf{O}_{\eta,z}\psi_2]\!] > t$ iff either $[\![\pi, \psi_2]\!] > \tau$, or both $[\![\pi, \psi_1]\!] > \tau$ and $[\![\pi^1, \psi_1\mathsf{O}_{\eta^{+1},z}\psi_2]\!] > t$. And indeed, 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 $\lim_{\eta(0)\to 0}\tau = \begin{cases} \infty & t > z \\ -\infty & t < z \end{cases}$. 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`, so 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 $\mathrm{LTL}^{\mathbf{disc}}[\mathcal{D}]$ 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, and is therefore not a legal discounting function. Still, the semantics of $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$ with these type of discounting functions is well defined. Moreover, consider an $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$ 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 - indeed, 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 $\eta(n) \leqslant \epsilon$.

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 LTL$^{\text{disc}}[\mathcal{D}]$ was taken in [63], where 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 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 (namely weighted average) renders the model-checking problem undecidable, while other, simpler operators (namely 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 LTL$^{\text{disc}}[\mathcal{D}]$ gives a logic, denoted LTL$^{\text{disc}\oplus}[\mathcal{D}]$, for which the satisfiability and model-checking problems are undecidable, both in their strict and non-strict versions. That is, for every $\sim \in \{<, \leqslant, =, \geqslant, >\}$, 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 LTL$^{\text{disc}\oplus}[\mathcal{D}]$ is undecidable, and then extract ideas from the proof, which are later used to show that the rest of the problems are undecidable. Our proofs apply to 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:

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

where $c \in \{x, y\}$ is a counter and $1 \leqslant i, j \leqslant n$ 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 $1 < i \leqslant m$, 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 $\text{DEC}(c)$ command, we assume that the machine never executes $\text{DEC}(c)$ 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 [59]. 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 $[\![\pi, \varphi]\!] \geqslant \frac{1}{2}$. 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 iff there exists a computation π such that $[\![\pi, \varphi]\!] \geqslant \frac{1}{2}$. The idea behind the construction is as follows. The formula φ is interpreted over computations over the atomic propositions $AP = \{1, ..., n, \#, x, y\}$, where $1, ..., n$ 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 6.1 *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), where 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 below, properties 1-3 can easily be specified by LTL formulas, such that computations which satisfy properties 1-3 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 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 $\frac{1}{2}$ iff $i = j$, 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 $AP = \{1, ..., n, \#, x, y\}$ (where $l_1, ..., l_n$ 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,

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

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

$$CheckFinal := \mathsf{G}((\bigvee_{i\in I} i) \to \mathsf{XG}\#) \wedge (\mathsf{F}(\bigvee_{i\in I} i))$$

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 $i \in \{1, ..., n\}$, the formula:

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

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

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

- If $l_i = \text{GOTO } l_j$, 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 $l_i = \text{IF } x{=}0 \text{ GOTO } l_j \text{ ELSE GOTO } l_k$ then $\psi_i := \mathsf{X}\left((x \to waitfor(k)) \wedge ((\neg x) \to waitfor(j))\right)$.
- If $l_i = \text{IF } y{=}0 \text{ GOTO } l_j \text{ ELSE GOTO } l_k$ then $\psi_i := \mathsf{X}\left((x\mathsf{U}y) \wedge waitfor(k)) \vee ((x\mathsf{U}\#) \wedge waitfor(j))\right)$.
- If $l_i = \text{HALT}$ we do not really need additional constraints, due to $CheckFinal$. Thus we have $\psi_i = \texttt{True}$.

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

[11] If $i = n$ then this line can be omitted from the initial machine, so w.l.o.g this does not happen.

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 $countX^{-1} = x\mathsf{U}_\eta\mathsf{X}\neg x$ 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

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

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, increase by 1, and decrease by 1 for each of the two counters. We define the formulas below for these tests. To explain the formulas, consider for example the formula $Comp(x, =)$. This formula compares the number of x's in the current configuration, with the number of x's in the next configuration. The comparison is based on the comparison we explained above, and is augmented by some parsing, as we need to reach the next configuration before comparing.

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

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

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

- If $l_i \in \big\{\text{GOTO } l_j, \text{IF } c{=}0 \text{ GOTO } l_j \text{ ELSE GOTO } l_k : c \in \{x,y\}\big\}$, then we need to make sure that the value 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 $\xi_i = \mathtt{True}$.

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 $[\![\pi,\varphi]\!] = \frac{1}{2}$. Indeed, 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 (indeed, 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 $[\![\pi, CheckCounters]\!] \geqslant \frac{1}{2}$, 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 $\mathcal{M}$, and 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 $[\![\pi,\varphi]\!] \geqslant \frac{1}{2}$ iff there exists a computation π such that $[\![\pi,\neg\varphi]\!] \leqslant \frac{1}{2}$, iff it is not true that $[\![\pi, \neg\varphi]\!] > \frac{1}{2}$ for every computation π. Thus, φ is $\frac{1}{2}$-co-valid iff $\neg\varphi$ 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 $[\![K, \varphi]\!] \geqslant v$ (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 6.2 *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* $\text{LTL}^{disc\oplus}[\mathcal{D}]$ *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 $\epsilon \geqslant \frac{1}{2}(\eta(i)-\eta(i+1))$ 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 non-strict model-checking problem, there does not seem to be an immediate reduction from the validity problem. Indeed, 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 $[\![\mathcal{K}, \varphi]\!] > v$ iff 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, and then $[\![\mathcal{K}, \varphi]\!] = v$. This convergence of the satisfaction value is at the heart of the problem, and avoiding it is the crux in our proofs.

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

Theorem 6.3 *The strict model-checking problem for* $\text{LTL}^{disc\oplus}[\mathcal{D}]$ *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 6.2, and consider the Kripke structure $\mathcal{K}$ that generates every computation. Observe that by Lemma 6.2 it holds that $[\![\mathcal{K}, \varphi]\!] \geqslant \frac{1}{2}$.

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 6.2 we conclude that $\mathcal{M}$ does not halt.

If $[\![\mathcal{K}, \varphi]\!] \leqslant \frac{1}{2}$, 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 $\epsilon = \inf(\eta(i) : i \text{ is the value of a counter in } \pi)$. 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 $[\![\pi, \varphi]\!] = 1$, so $[\![\pi, \xi]\!] = 1$.

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

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

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 $n \to \infty$, and 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 $\frac{1}{2}$, and 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 6.4 *The model-checking problem for* $\mathrm{LTL}^{disc\oplus}[\mathcal{D}]$ *is undecidable for every* $\mathcal{D} \neq \emptyset$.

Proof: Recall that for every Kripke structure $\mathcal{K}$ and formula φ it holds that $[\![\mathcal{K},\varphi]\!] \geqslant v$ iff 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 6.2 and the formula ψ such that for every computation π we have that $[\![\pi,\psi]\!] = \frac{1}{2}+\frac{1}{2}\epsilon$, where $\epsilon = \inf(\eta(i)-\eta(i+1) : i \text{ is the value of a counter in } \pi)$. 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 iff 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 $[\![\pi,\varphi]\!] = \frac{1}{2}$, and thus $[\![\pi,\neg\varphi]\!] = \frac{1}{2}$. Since the counters in π are bounded (as the run is halting), then $[\![\pi,\psi]\!] > \frac{1}{2}$, and thus $[\![\pi,\theta]\!] > \frac{1}{2}$.

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

- If π cheats in the commands, then $[\![\pi,\neg\varphi]\!] = 0$, so $[\![\pi,\theta]\!] = 0+\frac{1}{2}[\![\pi,\psi]\!] \leqslant \frac{1}{2}$.
- 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 $\epsilon \geqslant \frac{1}{2}(\eta(i)-\eta(i+1)) = \epsilon'$ for the smallest difference $\eta(i)-\eta(i+1)$ in π. Thus, $[\![\pi,\theta]\!] \leqslant \frac{1}{2}$.

□

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

Theorem 6.5 *For every $\sim\in\{<,\leqslant,=,\geqslant,>\}$ and for every $\mathcal{D}\neq\emptyset$, the following problems are undecidable.*

- *Model checking: given an* $\text{LTL}^{\textit{disc}\oplus}[\mathcal{D}]$ *formula φ, a system $\mathcal{K}$, and a threshold v, whether $[\![K,\varphi]\!] \sim v$.*
- *Satisfiability: given an* $\text{LTL}^{\textit{disc}\oplus}[\mathcal{D}]$ *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* $\text{LTL}^{\textit{disc}\oplus}[\mathcal{D}]$ *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 6.3, 6.4, 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 6.1,6.3, and 6.4 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's the complement of the model-checking for the "$\geqslant$" case.

- The satisfiability problem for the $<$ case is undecidable, since, by Observation 3, it is equivalent to 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 $\text{LTL}^{\text{disc}}[\mathcal{D}]$ and $\text{LTL}^{\triangledown}$

As shown in Section 6.1, adding the operator $\oplus$ to $\text{LTL}^{\text{disc}}[\mathcal{D}]$ 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 below that the logic $\text{LTL}^{\triangledown}$ (see Section 3.4) consists of such propositional quality operators.

As elaborated in Section 3.4, $\text{LTL}^{\triangledown}$ is a fragment of $\text{LTL}[\mathcal{F}]$ 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 $⧨_\lambda$ – 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 4.5.

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

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

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

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

- $\delta(⧩_\lambda \varphi > t, \sigma) = \begin{cases} \delta(\varphi > \frac{2t+\lambda-1}{2\lambda}, \sigma) & \text{if } \frac{t}{\lambda} < 1, \\ \texttt{False} & \text{if } \frac{2t+\lambda-1}{2\lambda} \geqslant 1, \end{cases}$

- $\delta(⧩_\lambda \varphi < t, \sigma) = \begin{cases} \delta(\varphi < \frac{2t+\lambda-1}{2\lambda}, \sigma) & \text{if } \frac{2t+\lambda-1}{2\lambda} \leqslant 1, \\ \texttt{True} & \text{if } \frac{2t+\lambda-1}{2\lambda} > 1. \end{cases}$

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

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).

In the propositional-quality front, our scheme, denoted $\text{LTL}[\mathcal{F}]$, 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 one hand, $\text{LTL}[\mathcal{F}]$ formulas can have exponentially many satisfaction values, which allows to succinctly represent and rank many ways of satisfying a specification, and on the other hand, model checking $\text{LTL}[\mathcal{F}]$ can be done in PSPACE, as the complexity for standard LTL.

On the temporal-quality front, our scheme, denoted $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$, is based on augmenting the *until* operator of LTL with an arbitrary set of discounting functions. This enables 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 the complexity.

7.2 Future Research

The expressive power of $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$ and $\mathrm{PLTL}^{\mathrm{disc}}[\mathcal{D}]$ In Section 5.2 we add discounting past operators to $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$, and show that model-checking can be solved with the same complexity as model-checking $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$. In the Boolean setting, it is known that past operators do not add expressive power to LTL (but do allow exponentially more succinct formulas) [56]. 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 $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$ and $\mathrm{PLTL}^{\mathrm{disc}}[\mathcal{D}]$. We conjecture that $\mathrm{PLTL}^{\mathrm{disc}}[\mathcal{D}]$ is strictly more expressive than $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$, and specifically, that the formula $\mathsf{F}_\eta\neg(\mathtt{True}\mathsf{S}_\eta p)$ over the atomic proposition $AP = \{p\}$ does not have an equivalent $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$ formula.

Fragments of $\mathrm{LTL}[\mathcal{F}]$ and $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$ 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 which is very expressive is GR(1) [64]. For practical applications, it would be useful to look at this fragment in the context of $\mathrm{LTL}[\mathcal{F}]$ and $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$.

For $\mathrm{LTL}[\mathcal{F}]$, 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 $\mathrm{LTL}[\mathcal{F}]$. In particular, the $\mathrm{LTL}[\mathcal{F}]$ analogue of the $\mathrm{GR}(1)$ fragment allows ranking of the Streett conditions. That is, instead of considering a formula of the form $(\mathsf{GF}\varphi_1 \wedge \ldots \wedge \mathsf{GF}\varphi_n) \to (\mathsf{GF}\psi_1 \wedge \ldots \wedge \mathsf{GF}\psi_n)$, we consider the formula $(\mathsf{GF}\varphi_1 \wedge \ldots \wedge \mathsf{GF}\varphi_n) \to f(\mathsf{GF}\psi_1, \ldots, \mathsf{GF}\psi_n)$, where f is a function that ranks different subsets of its inputs. This allows us to prioritize the importance of the goals $\psi_1, ..., \psi_n$.

For $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$, it is less clear that standard GR(1) algorithms can readily be adapted, and this is an important direction for future work. The benefit of using $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$ 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 $(\mathsf{GF}\varphi_1 \wedge \ldots \wedge \mathsf{GF}\varphi_n) \to (\mathsf{GF}\psi_1 \wedge \ldots \wedge \mathsf{GF}\psi_n)$ we can use the formula $(\mathsf{GF}_\eta\varphi_1 \wedge \ldots \wedge \mathsf{GF}_\eta\varphi_n) \to (\mathsf{GF}_\eta\psi_1 \wedge \ldots \wedge \mathsf{GF}_\eta\psi_n)$, 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 which is especially relevant for $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$ 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 $\mathrm{LTL}^{\mathrm{disc}}[\mathcal{D}]$ will also entail simplifications in our constructions. In particular, observe that the construction in Lemma 4.10 is exponential in the nesting depth of the formula (but also in the description of the discounting factors) and thus if the nesting depth is bounded, the construction becomes much smaller.

References

[1] E. Allender, P. Bürgisser, J. Kjeldgaard-Pedersen, and P.B. Miltersen. On the complexity of numerical analysis. *SIAM J. Comput.*, 38(5):1987–2006, 2009.

[2] S. Almagor, G. Avni, and O. Kupferman. Automatic generation of quality specifications. In *25th CAV*, volume 8044 of *Lecture Notes in Computer Science*, pages 479–494, 2013.

[3] S. Almagor, U. Boker, and O. Kupferman. What's decidable about weighted automata? In *9th Int. Symp. on Automated Technology for Verification and Analysis*, volume 6996 of *Lecture Notes in Computer Science*, pages 482–491. Springer, 2011.

[4] S. Almagor, U. Boker, and O. Kupferman. Formalizing and reasoning about quality. In *Proc. 40th Int. Colloq. on Automata, Languages, and Programming*, volume 7966 of *Lecture Notes in Computer Science*, pages 15 – 27. Springer, 2013.

[5] S. Almagor, U. Boker, and O. Kupferman. Discounting in LTL. In *Proc. 20th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, volume 8413 of *Lecture Notes in Computer Science*, pages 424–439. Springer, 2014.

[6] S. Almagor, Y. Hirshfeld, and O. Kupferman. Promptness in omega-regular automata. In *8th Int. Symp. on Automated Technology for Verification and Analysis*, volume 6252, pages 22–36. Springer, 2010.

[7] S. Almagor and O. Kupferman. Max and sum semantics for alternating weighted automata. In *9th Int. Symp. on Automated Technology for Verification and Analysis*, volume 6996 of *Lecture Notes in Computer Science*, pages 13–27. Springer, 2011.

[8] S. Almagor and O. Kupferman. High-quality synthesis against stochastic environments. Submitted, 2015.

[9] S. Arora and B. Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009.

[10] R. Bloem, K. Chatterjee, T.A. Henzinger, and B. Jobstmann. Better quality in synthesis through quantitative objectives. In *Proc. 21st Int. Conf. on Computer Aided Verification*, volume 5643 of *Lecture Notes in Computer Science*, pages 140–156. Springer, 2009.

[11] A. Bohy, V. Bruyère, E. Filiot, and J-F. Raskin. Synthesis from LTL specifications with mean-payoff objectives. In *Proc. 19th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, volume 7795 of *Lecture Notes in Computer Science*, pages 169–184. Springer, 2013.

[12] M. Bojańczyk and T. Colcombet. Bounds in ω-regularity. In *Proc. 21st IEEE Symp. on Logic in Computer Science*, pages 285–296, 2006.

[13] U. Boker, K. Chatterjee, T.A. Henzinger, and O. Kupferman. Temporal specifications with accumulative values. *ACM Transactions on Computational Logic*, 15(4):27:1–27:25, 2014.

[14] U. Boker, T.A. Henzinger, and J. Otop. The target discounted-sum problem. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, pages 750–761, 2015.

[15] U. Boker, O. Kupferman, and A. Rosenberg. Alternation removal in Büchi automata. In *Proc. 37th Int. Colloq. on Automata, Languages, and Programming*, volume 6199, pages 76–87, 2010.

[16] P. Bouyer, N. Markey, and R. Matteplackel. Averaging in LTL. In *CONCUR 2014 - Concurrency Theory - 25th International Conference, CONCUR 2014, Rome, Italy, September 2-5, 2014. Proceedings*, pages 266–280, 2014.

[17] G. Bruns and P. Godefroid. Model checking with multi-valued logics. In *Proc. 31st Int. Colloq. on Automata, Languages, and Programming*, volume 3142 of *Lecture Notes in Computer Science*, pages 281–293, 2004.

[18] P. Černỳ, K. Chatterjee, T.A. Henzinger, A. Radhakrishna, and R. Singh. Quantitative synthesis for concurrent programs. In *Proc. 23rd Int. Conf. on Computer Aided Verification*, pages 243–259, 2011.

[19] K. Chatterjee, V. Forejt, and D. Wojtczak. Multi-objective discounted reward verification in graphs and mdps. In *Logic for Programming, Artificial Intelligence, and Reasoning - 19th International Conference, LPAR-19, Stellenbosch, South Africa, December 14-19, 2013. Proceedings*, pages 228–242, 2013.

[20] E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[21] M. Dam. CTL* and ECTL* as fragments of the modal μ-calculus. *Theoretical Computer Science*, 126:77–96, 1994.

[22] L. De Alfaro, M. Faella, T.A. Henzinger, R. Majumdar, and M. Stoelinga. Model checking discounted temporal properties. *Theoretical Computer Science*, 345(1):139–170, 2005.

[23] L. De Alfaro, M. Faella, and M. Stoelinga. Linear and branching metrics for quantitative transition systems. In *Proc. 31st Int. Colloq. on Automata, Languages, and Programming*, pages 97–109, 2004.

[24] L. De Alfaro, T.A. Henzinger, and R. Majumdar. Discounting the future in systems theory. In *Proc. 30th Int. Colloq. on Automata, Languages, and Programming*, volume 2719 of *Lecture Notes in Computer Science*, pages 1022–1037, 2003.

[25] J. Desharnais, V. Gupta, R. Jagadeesan, and P. Panangaden. Metrics for labelled markov processes. *Theoretical Computer Science*, 318(3):323–354, 2004.

[26] A. Donzé and O. Maler. 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, pages 92–106, 2010.

[27] A. Donzé, O. Maler, E. Bartocci, D. Nickovic, R. Grosu, and S. Smolka. On temporal logic and signal processing. In *Proceedings of the 10th International Conference on Automated Technology for Verification and Analysis*, ATVA'12, pages 92–106. Springer-Verlag, 2012.

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

[29] M. Droste, W. Kuich, and G. Rahonis. Multi-valued MSO logics overwords and trees. *Fundam. Inform.*, 84(3-4):305–327, 2008.

[30] M. Droste and G. Rahonis. Weighted automata and weighted logics with discounting. *Theoretical Computer Science*, 410(37):3481–3494, 2009.

[31] M. Droste and H. Vogler. Weighted automata and multi-valued logics over arbitrary bounded lattices. *Theoretical Computer Science*, 418:14–36, 2012.

[32] E.A. Emerson and J.Y. Halpern. Sometimes and not never revisited: On branching versus linear time. *Journal of the ACM*, 33(1):151–178, 1986.

[33] E.A. Emerson and C-L. Lei. Modalities for model checking: Branching time logic strikes back. In *Proc. 12th ACM Symp. on Principles of Programming Languages*, pages 84–96, 1985.

[34] E.A. Emerson and C-L. Lei. Efficient model checking in fragments of the propositional μ-calculus. In *Proc. 1st IEEE Symp. on Logic in Computer Science*, pages 267–278, 1986.

[35] M. Faella, A. Legay, and M. Stoelinga. Model checking quantitative linear time logic. *Electr. Notes Theor. Comput. Sci.*, 220(3):61–77, 2008.

[36] E. Filiot, R. Gentilini, and J-F. Raskin. Finite-valued weighted automata. In *34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2014, December 15-17, 2014, New Delhi, India*, pages 133–145, 2014.

[37] M. Fränzle, M.R. Hansen, and H. Ody. Discounted duration calculus. 2015.

[38] P. Gastin and D. Oddoux. Fast LTL to Büchi automata translation. In *Proc. 13th Int. Conf. on Computer Aided Verification*, volume 2102 of *Lecture Notes in Computer Science*, pages 53–65. Springer, 2001.

[39] Y. Gurevich and L. Harrington. Trees, automata, and games. In *Proc. 14th ACM Symp. on Theory of Computing*, pages 60–65. ACM Press, 1982.

[40] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6:102–111, 1994.

[41] G.J. Holzmann. *The Spin Model Checker:primer and reference manual*. Addison-Wesley, 2004.

[42] IEEE. IEEE standard multivalue logic system for VHDL model interoperability (Std_logic_1164), 1993.

[43] S.H. Kan. *Metrics and Models in Software Quality Engineering*. Addison-Wesley Longman Publishing Co., Inc., 2nd edition, 2002.

[44] D. Kirsten and S. Lombardy. Deciding unambiguity and sequentiality of polynomially ambiguous min-plus automata. In *26th International Symposium on Theoretical Aspects of Computer Science, STACS 2009, February 26-28, 2009, Freiburg, Germany, Proceedings*, pages 589–600, 2009.

[45] D. Krob. The equality problem for rational series with multiplicities in the tropical semiring is undecidable. *International Journal of Algebra and Computation*, 4(3):405–425, 1994.

[46] O. Kupferman. Sanity checks in formal verification. In *Proc. 17th Int. Conf. on Concurrency Theory*, volume 4137 of *Lecture Notes in Computer Science*, pages 37–51. Springer, 2006.

[47] O. Kupferman and Y. Lustig. Lattice automata. In *Proc. 8th Int. Conf. on Verification, Model Checking, and Abstract Interpretation*, volume 4349 of *Lecture Notes in Computer Science*, pages 199 – 213. Springer, 2007.

[48] O. Kupferman, N. Piterman, and M.Y. Vardi. Safraless compositional synthesis. In *Proc. 18th Int. Conf. on Computer Aided Verification*, volume 4144 of *Lecture Notes in Computer Science*, pages 31–44. Springer, 2006.

[49] O. Kupferman, N. Piterman, and M.Y. Vardi. From liveness to promptness. *Formal Methods in System Design*, 34(2):83–103, 2009.

[50] O. Kupferman and M.Y. Vardi. Synthesis with incomplete informatio. In *2nd Int. Conf. on Temporal Logic*, pages 91–106, 1997.

[51] O. Kupfermann and M.Y. Vardi. Safraless decision procedures. In *Proc. 46th IEEE Symp. on Foundations of Computer Science*, pages 531–540, 2005.

[52] O. Kupferman, M.Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *Journal of the ACM*, 47(2):312–360, 2000.

[53] R.P. Kurshan. *FormalCheck User's Manual.* Cadence Design, Inc., 1998.

[54] M.Z. Kwiatkowska. Quantitative verification: models techniques and tools. In *ESEC/SIGSOFT FSE*, pages 449–458, 2007.

[55] F. Laroussinie and Ph. Schnoebelen. A hierarchy of temporal logics with past. In *Proc. 11th Symp. on Theoretical Aspects of Computer Science*, pages 47–58, 1994.

[56] O. Lichtenstein, A. Pnueli, and L. Zuck. The glory of the past. In *Logics of Programs*, volume 193 of *Lecture Notes in Computer Science*, pages 196–218. Springer, 1985.

[57] E. Mandrali. Weighted ltl with discounting. In *CIAA*, volume 7381 of *Lecture Notes in Computer Science*, pages 353–360, 2012.

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

[59] M.L. Minsky. *Computation: Finite and Infinite Machines*. Prentice Hall, 1 edition, 1967.

[60] S. Miyano and T. Hayashi. Alternating finite automata on ω-words. *Theoretical Computer Science*, 32:321–330, 1984.

[61] M. Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311, 1997.

[62] S. Moon, K.H. Lee, and D. Lee. Fuzzy branching temporal logic. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 34(2):1045–1055, 2004.

[63] S. Nakagawa and I. Hasuo. Near-optimal scheduler synthesis for LTL with future discounting. In *10th International Symposium on Trustworthy Global Computing*, 2015.

[64] N. Piterman, A. Pnueli, and Y. Saar. Synthesis of reactive(1) designs. In *Proc. 7th Int. Conf. on Verification, Model Checking, and Abstract Interpretation*, volume 3855 of *Lecture Notes in Computer Science*, pages 364–380. Springer, 2006.

[65] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. 16th ACM Symp. on Principles of Programming Languages*, pages 179–190, 1989.

[66] C.E. Shannon. The synthesis of two terminal switching circuits. 28(1):59–98, 1949.

[67] L.S. Shapley. Stochastic games. In *Proc. of the National Academy of Science*, volume 39, page 1095. National Academy of Sciences, 1953.

[68] D. Spinellis. *Code Reading: The Open Source Perspective*. Addison-Wesley, 2003.

[69] W. Thomas. Automata on infinite objects. *Handbook of Theoretical Computer Science*, 2:133–191, 1990.

[70] M.Y. Vardi. An automata-theoretic approach to linear temporal logic. In F. Moller and G. Birtwistle, editors, *Logics for Concurrency: Structure versus Automata*, volume 1043 of *Lecture Notes in Computer Science*, pages 238–266. Springer, 1996.

[71] M.Y. Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification. In *Proc. 1st IEEE Symp. on Logic in Computer Science*, pages 332–344, 1986.

[72] M.Y. Vardi and Pierre Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.

Latticed-LTL Synthesis in the Presence of Noisy Inputs[*]

Shaull Almagor[†] *Orna Kupferman* [‡]

Abstract

In the classical *synthesis* problem, we are given a specification ψ over sets of input and output signals, and we synthesize a finite-state transducer that realizes ψ: with every sequence of input signals, the transducer associates a sequence of output signals so that the generated computation satisfies ψ. In recent years, researchers consider extensions of the classical Boolean setting to a multi-valued one. We study a multi-valued setting in which the truth values of the input and output signals are taken from a finite lattice, and so is the satisfaction value of specifications. We consider specifications in *latticed linear temporal logic* (LLTL). In LLTL, conjunctions and disjunctions correspond to the meet and join operators of the lattice, respectively, and the satisfaction values of formulas are taken from the lattice too. The lattice setting arises in practice, for example in specifications involving priorities or in systems with inconsistent viewpoints.

We solve the LLTL synthesis problem, where the goal is to synthesize a transducer that realizes the given specification in a desired satisfaction value.

For the classical synthesis problem, researchers have studied a setting with incomplete information, where the truth values of some of the input signals are hidden and the transducer should nevertheless realize ψ. For the multi-valued setting, we introduce and study a new type of incomplete information, where the truth values of some of the input signals may be noisy, and the transducer should still realize ψ in the desired satisfaction value. We study the problem of noisy LLTL synthesis, as well as the theoretical aspects of the setting, like the amount of noise a transducer may tolerate, or the effect of perturbing input signals on the satisfaction value of a specification.

1 Introduction

Synthesis is the automated construction of a system from its specification. The basic idea is simple and appealing: instead of developing a system and verifying that it adheres to its

[*] Published in the proceeding of the 17th Foundations of Software Science and Computation Structures International Conference, LNCS 8412, pages 226–241, Springer, 2014. A full version was submitted.

[†] The Hebrew University, Jerusalem, Israel.

[‡] The Hebrew University, Jerusalem, Israel.

specification, we would like to have an automated procedure that, given a specification, constructs a system that is correct by construction. The first formulation of synthesis goes back to Church [10]. The modern approach to synthesis was initiated by Pnueli and Rosner, who introduced LTL (linear temporal logic) synthesis [25]: We are given an LTL formula ψ over sets I and O of input and output signals, and we synthesize a finite-state system that *realizes* ψ. At each moment in time, the system reads a truth assignment, generated by the environment, to the signals in I, and it generates a truth assignment to the signals in O. Thus, with every sequence of inputs, the transducer associates a sequence of outputs, and it ψ if all the computations that are generated by the interaction satisfy ψ. Synthesis has attracted a lot of research and interest [30].

In recent years, researchers have considered extensions of the classical Boolean setting to a multi-valued one, where the atomic propositions are multi-valued, and so is the satisfaction value of specifications. The multi-valued setting arises directly in systems in which the designer can give to the atomic propositions rich values, expressing, for example, energy consumption, waiting time, or different levels of confidence [5, 1], and arises indirectly in probabilistic settings, systems with multiple and inconsistent view-points, specifications with priorities, and more [21, 14, 2]. Adjusting the synthesis problem to this setting, one works with multi-valued specification formalisms. In such formalisms, a specification ψ maps computations in which the atomic propositions take values from a domain D to a satisfaction value in D. For example, ψ may map a computation in $(\{0, 1, 2, 3\}^{\{p\}})^{\omega}$ to the maximal value assigned to the (multi-valued) atomic proposition p during the computation. Accordingly, the synthesis problem in the multi-valued setting gets as input a specification ψ and a predicate $P \subseteq D$ of desired values, and seeks a system that reads assignments in D^I, responds with assignments in D^O, and generates only computations whose satisfaction value is in P. The synthesis problem has been solved for several multi-valued settings [3, 4, 1].

A different extension of the classical setting of synthesis considers settings in which the system has *incomplete information* about its environment. Early work on incomplete information considers settings in which the system can read only a subset of the signals in I and should still generate only computations that satisfy the specification, which refers to all the signals in $I \cup O$ [16, 26, 19, 6, 7]. The setting is equivalent to a game with incomplete information, extensively studied in [27]. As shown there, the common practice in handling incomplete information is to move to an exponentially-larger game of complete information, where each state corresponds to a set of states that are indistinguishable by a player with incomplete information in the original game.

More recent work on synthesis with incomplete information studies richer types of incomplete information. In [8], the authors study a setting in which the transducer can read some of the input signals some of the time. In more detail, *sensing* the truth value of an input signal has a cost, the system has a budget for sensing, and it tries to realize the specification while minimizing the required sensing budget. In [32], the authors study games with *errors*. Such games correspond to a synthesis setting in which there are positions during the interaction in which input signals are read by the system with an error. The games are characterized by the number or rate of errors that the system has to cope with, and by the ability of the system to detect whether a current input is erred.

In this work we introduce and study a different model of incomplete information in the multi-valued setting. In our model, the system always reads all input signals, but their value may be perturbed according to a known noise function. This setting naturally models incomplete information in real-life multi-valued settings. For example, when the input is read by sensors that are not accurate (e.g., due to bounded precision, or to probabilistic measuring) or when the input is received over a noisy channel and may come with some distortion. The multi-valued setting we consider is that of *finite lattices*. A lattice is a partially-ordered set $\mathcal{L} = \langle A, \leqslant \rangle$ in which every two elements ℓ and ℓ' have a least upper bound (ℓ *join* ℓ', denoted $\ell \vee \ell'$) and a greatest lower bound (ℓ *meet* ℓ', denoted $\ell \wedge \ell'$). Of special interest are two classes of lattices: (1) Fully ordered, where $\mathcal{L} = \langle \{1, \ldots, n\}, \leqslant \rangle$, for an integer $n \geqslant 1$ and the usual "less than or equal" order. In this lattice, the operators $\vee$ and $\wedge$ correspond to $\max$ and $\min$, respectively. (2) Power-set lattices, where $\mathcal{L} = \langle 2^X, \subseteq \rangle$, for a finite set X, and the containment (partial) order. In this lattice, the operators $\vee$ and $\wedge$ correspond to $\cup$ and $\cap$, respectively.

The lattice setting is a good starting point to the multi-valued setting. While their finiteness circumvents the infinite-state space of dense multi-values, lattices are sufficiently rich to capture many quantitative settings. Fully-ordered lattices are sometimes useful as is (for example, when modeling priorities [2]), and sometimes thanks to the fact that real values can often be abstracted to finitely many linearly ordered classes. The power-set lattice models a wide range of partially-ordered values. For example, in a setting with inconsistent viewpoints, we have a set X of agents, each with a different viewpoint of the system, and the truth value of a signal or a formula indicates the set of agents according to whose viewpoint the signal or the formula are true. As another example, in a peer-to-peer network, one can refer to the different attributes of the communication channels by assigning with them subsets of attributes. From a technical point of view, the fact that lattices are partially ordered poses challenges that do not exist in (finite and infinite) full orders. For example, as we are going to see, the fact that a specification is realizable with value ℓ and with value ℓ' does not imply it is realizable with value $\ell \vee \ell'$, which trivially holds for full orders.

We start by defining lattices and the logic *Latticed LTL* (LLTL, for short). We then study theoretical properties of LLTL: We study cases where the set of attainable truth values of an LLTL formula are closed under $\vee$, thus a maximal attainable value exists, even when the lattice elements are partially ordered. We also study stability properties, namely the affect of perturbing the values of the atomic propositions on the satisfaction value of formulas. We continue to the synthesis and the noisy-synthesis problems for LLTL, which we solve via a translation of LLTL formulas to Boolean automata. We show that by working with universal automata, we can handle the exponential blow-up that incomplete information involves together with the exponential blow-up that determination (or alternation removal, if we take a Safraless approach) involves, thus the noisy-synthesis problem stays 2EXPTIME-complete, as it is for LTL.

2 Preliminaries

2.1 Lattices

Consider a set A, a partial order $\leqslant$ on A, and a subset P of A. An element $\ell \in A$ is an *upper bound* on P if $\ell \geqslant \ell'$ for all $\ell' \in P$. Dually, ℓ is a *lower bound* on P if $\ell \leqslant \ell'$ for all $\ell' \in P$. The pair $\langle A, \leqslant \rangle$ is a *lattice* if for every two elements $\ell, \ell' \in A$, both the least upper bound and the greatest lower bound of $\{\ell, \ell'\}$ exist, in which case they are denoted $\ell \vee \ell'$ (ℓ *join* ℓ') and $\ell \wedge \ell'$ (ℓ *meet* ℓ'), respectively. We use $\ell < \ell'$ to indicate that $\ell \leqslant \ell'$ and $\ell \neq \ell'$. We say that ℓ is a *child* of ℓ', denoted $\ell \prec \ell'$, if $\ell < \ell'$ and there is no ℓ'' such that $\ell < \ell'' < \ell'$.

A lattice $\mathcal{L} = \langle A, \leqslant \rangle$ is *finite* if A is finite. Note that finite lattices are *complete*: every subset of A has a least-upper and a greatest-lower bound. We use $\top$ (*top*) and $\bot$ (*bottom*) to denote the least-upper and greatest-lower bounds of A, respectively. A lattice is *distributive* if for every $\ell_1, \ell_2, \ell_3 \in A$, we have $\ell_1 \wedge (\ell_2 \vee \ell_3) = (\ell_1 \wedge \ell_2) \vee (\ell_1 \wedge \ell_3)$ and $\ell_1 \vee (\ell_2 \wedge \ell_3) = (\ell_1 \vee \ell_2) \wedge (\ell_1 \vee \ell_3)$. The traditional disjunction and conjunction logic operators correspond to the join and meet lattice operators. In a general lattice, however, there is no natural counterpart to negation. A *De-Morgan* (or *quasi-Boolean*) lattice is a lattice in which every element a has a unique complement element $\neg\ell$ such that $\neg\neg\ell = \ell$, De-Morgan rules hold, and $\ell \leqslant \ell'$ implies $\neg\ell' \leqslant \neg\ell$. In the rest of this paper we consider only finite distributive De-Morgan lattices. We focus on two classes of such lattices: (1) Fully ordered, where $\mathcal{L} = \langle \{1, \ldots, n\}, \leqslant \rangle$, for an integer $n \geqslant 1$ and the usual "less than or equal" order. Note that in this lattice, the operators $\vee$ and $\wedge$ correspond to $\max$ and $\min$, respectively, and $\neg i = n - i + 1$. (2) Power-set lattices, where $\mathcal{L} = \langle 2^X, \subseteq \rangle$, for a finite set X, and the containment (partial) order. Note that in this lattice, the operators $\vee$ and $\wedge$ correspond to $\cup$ and $\cap$, respectively, and negation corresponds to complementation.

Consider a lattice $\mathcal{L} = \langle A, \leqslant \rangle$. A *join irreducible* element in $\mathcal{L}$ is $l \in A$ such that $l \neq \bot$ and for all elements $l_1, l_2 \in A$, if $l_1 \vee l_2 \geqslant l$, then $l_1 \geqslant l$ or $l_2 \geqslant l$. For example, the join irreducible elements in $\langle 2^X, \subseteq \rangle$ are all singletons $\{x\}$, for $x \in X$. By *Birkhoff's representation theorem* for finite distributive lattices, in order to prove that $l_1 = l_2$, it is sufficient to prove that for every join irreducible element l it holds that $l_1 \geqslant l$ iff $l_2 \geqslant l$. We denote the set of join irreducible elements of $\mathcal{L}$ by $\mathrm{JI}(\mathcal{L})$. For convenience, we often talk about a lattice $\mathcal{L}$ without specifying A and $\leqslant$. We then abuse notations and refer to $\mathcal{L}$ as a set of elements and talk about $l \in \mathcal{L}$ or about assignments in $\mathcal{L}^{AP}$ (rather than $l \in A$ or assignments in A^{AP}).

2.2 The logic LLTL

The logic LLTL is a natural generalization of LTL to a multi-valued setting, where the atomic propositions take values from a lattice $\mathcal{L}$ [9, 17]. Given a (finite distributive De-Morgan) lattice $\mathcal{L}$, the syntax of LLTL is given by the following grammar, where p ranges over a set AP of atomic propositions, and ℓ ranges over $\mathcal{L}$.

$$\varphi := \ell \mid p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathsf{X}\varphi \mid \varphi\mathsf{U}\varphi.$$

The semantics of LLTL is defined with respect to a *computation* $\pi = \pi_0, \pi_1, \ldots \in (\mathcal{L}^{AP})^\omega$. Thus, in each moment in time the atomic propositions get values from $\mathcal{L}$. Note that classical LTL coincides with LLTL defined with respect to the two-element fully-ordered lattice. For a position $i \geqslant 0$, we use π^i to denote the suffix $\pi_i, \pi_{i+1}, \ldots$ of π. Given a computation π and an LLTL formula φ, the *satisfaction value* of φ in π, denoted $[\![\pi, \varphi]\!]$, is defined by induction on the structure of φ as follows (the operators on the right-hand side are the join, meet, and complementation operators of $\mathcal{L}$).

- $[\![\pi, \ell]\!] = \ell.$
- $[\![\pi, \varphi \vee \psi]\!] = [\![\pi, \varphi]\!] \vee [\![\pi, \psi]\!].$
- $[\![\pi, p]\!] = \pi_0(p).$
- $[\![\pi, \mathsf{X}\varphi]\!] = [\![\pi^1, \varphi]\!].$
- $[\![\pi, \neg\varphi]\!] = \neg[\![\pi, \varphi]\!].$
- $[\![\pi, \varphi\mathsf{U}\psi]\!] = \bigvee_{i \geqslant 0}([\![\pi^i, \psi]\!] \wedge \bigwedge_{0 \leqslant j < i}[\![\pi^j, \varphi]\!]).$

Example 2.1 *Consider a setting in which three agents $a, b,$ and c have different view-points on a system $\mathcal{S}$. A truth assignment for the atomic propositions is then a function in $(2^{\{a,b,c\}})^{AP}$ assigning to each $p \in AP$ the set of agents according to whose view-point p is true. We reason about $\mathcal{S}$ using the lattice $\mathcal{L} = \langle 2^{\{a,b,c\}}, \subseteq \rangle$. For example, the truth value of the formula $\psi = G(req \to F grant)$ in a computation is the set of agents according to whose view-point, whenever a request is sent, it is eventually granted.*

Remark 2.1 Recall that the constants `True` and `False` in LTL do not add to its expressive power. Indeed, `True` can be replaced by $p \vee (\neg p)$, for a Boolean atomic proposition p, and similarly for `False`. This is not the case for the constants $\ell \in \mathcal{L}$ in LLTL. For example, consider the linear lattice $\langle \{1, ..., 5\}, \leqslant \rangle$. It is easy to show that for every formula φ (without constant) over the atomic propositions AP, the computation π for which $\pi^i(p) = 3$ for every $i \geqslant 0$ and $p \in AP$, satisfies $[\![\pi, \varphi]\!] = 3$. Thus, for example, there is no formula φ that is equivalent to the constant 1.

Constants can be used to upper or lower bound the satisfaction value of an LLTL formula. For example, the truth value of the LLTL formula $\{a, b\} \wedge \psi$, defined with respect to the lattice $\langle 2^{\{a,b,c\}}, \subseteq\rangle$, is the set of agents that is both a subset of $\{a, b\}$ and according to whose viewpoint, the specification ψ is satisfied.

2.3 LLTL **synthesis**

Consider a lattice $\mathcal{L}$ and finite disjoint sets I and O of input and output signals that take values in $\mathcal{L}$. An (I/O)*-transducer over* $\mathcal{L}$ models an interaction between an environment that generates in each moment in time an input in $\mathcal{L}^I$ and a system that responds with outputs in L^O. Formally, an (I/O)-transducer over $\mathcal{L}$ (*transducer*, when I, O, and $\mathcal{L}$ are clear from the context) is a tuple $\mathcal{T} = \langle \mathcal{L}, I, O, S, s_0, \eta, \tau\rangle$ where S is a finite set of states, $s_0 \in S$ is an initial state, $\eta : S \times \mathcal{L}^I \to S$ is a deterministic transition function, and $\tau : S \to \mathcal{L}^O$ is a labeling function. We extend η to words in $(\mathcal{L}^I)^*$ in the straightforward way. Thus, $\eta : (\mathcal{L}^I)^* \to S$ is such that $\eta(\epsilon) = s_0$, and for $x \in (\mathcal{L}^I)^*$ and $i \in \mathcal{L}^I$, we have $\eta(x \cdot i) = \eta(\eta(x), i)$. Each transducer $\mathcal{T}$ induces a *strategy* $f_{\mathcal{T}} : (\mathcal{L}^I)^* \to \mathcal{L}^O$ where for all $w \in (\mathcal{L}^I)^*$, we have $f_{\mathcal{T}}(w) = \tau(\eta(w))$. Thus, $f_{\mathcal{T}}(w)$ is the letter that $\mathcal{T}$ outputs after reading the sequence w of input letters. Given a sequence $i_0, i_1, i_2, \ldots \in (\mathcal{L}^I)^\omega$ of input assignments, the transducer generates the computation $\rho = (i_0 \cup o_0), (i_1 \cup o_1), (i_2 \cup o_2), \ldots \in (\mathcal{L}^{I \cup O})^\omega$, where for all $j \geqslant 1$, we have $o_j = f_{\mathcal{T}}(i_0 \cdots i_{j-1})$.

Consider a lattice $\mathcal{L}$, an LLTL formula φ over the atomic propositions $I \cup O$, and a predicate $P \subseteq \mathcal{L}$. We say that a transducer $\mathcal{T}$ *realizes* $\langle \varphi, P\rangle$ if for every computation ρ of $\mathcal{T}$, it holds that $[\![\rho, \varphi]\!] \in P$. The *realizability* problem for LLTL is to determine, given φ and P, whether there exists a transducer that realizes $\langle \varphi, P\rangle$. We then say that φ is (I/O)-realizable with values in P. The *synthesis* problem is then to generate such a transducer. Of special interest are predicates P that are upward closed. Thus, P is such that for all $\ell \in \mathcal{L}$, if $\ell \in P$ then $\ell' \in P$ for all $\ell' \geqslant \ell$.

2.4 Noisy synthesis

Consider an LLTL formula φ over atomic proposition $I \cup O$ and a predicate P. In *noisy synthesis*, we consider the synthesis problem in a setting in which the inputs are read with some perturbation and the goal is to synthesize a transducer that nevertheless realizes $\langle \varphi, P\rangle$.

In order to formalize the above intuition, we first formalize the notion of noise. Consider a lattice $\mathcal{L} = \langle A, \leqslant\rangle$ and two elements $\ell_1, \ell_2 \in \mathcal{L}$. We define the *distance* between ℓ_1 and ℓ_2, denoted $d(\ell_1, \ell_2)$, as the shortest path from ℓ_1 to ℓ_2 in the undirected graph $\langle A, E_<\rangle$ in which $E_<(v, v')$ iff $v < v'$ or $v' < v$. For example, in the fully-ordered lattice $\mathcal{L}$, we have $d(i, j) = |i - j|$, and in the power-set lattice, the distance coincides with the Hamming distance, thus $d(X_1, X_2) = |(X_1 \backslash X_2) \cup (X_2 \backslash X_1)|$. For two assignments $f, f' \in \mathcal{L}^{AP}$, we define $d(f, f') = \max_{p \in AP} d(f(p), f'(p))$.

We assume we are given a *noise function* $\nu : \mathcal{L}^I \to 2^{\mathcal{L}^I}$, describing the possible perturbations of each input. That is, for every $i \in \mathcal{L}^I$ the set $\nu(i)$ consists of the inputs that may have been actually generated by the environment, when the system reads i. A natural noise function is $\nu(i) = \{j : d(i, j) \leqslant \gamma\}$, for some constant γ, which is the γ-units ball around i. Given a noise function ν and two computations $\pi, \pi' \in (\mathcal{L}^{I \cup O})^\omega$, we say that π' is *ν-indistinguishable* from π if for every $i \geqslant 0$, we have that $\pi'_i|_I \in \nu(\pi_i|_I)$ and $\pi'_i|_O = \pi_i|_O$, where $\sigma|_I$ is the restriction of $\sigma \in \mathcal{L}^{I \cup O}$ to inputs in I, and similarly for $\sigma|_O$ and O. Thus, π' is obtained from π by changing only the assignment to input signals, within ν. Note that ν need not be a symmetric function, nor is the definition of ν-indistinguishablity. We say that a transducer $\mathcal{T}$ *realizes $\langle \varphi, P \rangle$ with noise ν* if for every computation π of $\mathcal{T}$, we have that $[\![\pi', \varphi]\!] \in P$ for all computations π' that are ν-indistinguishable from π. Thus, the reaction of $\mathcal{T}$ on every input sequence satisfies φ in a desired satisfaction value even if the input sequence is read with noise ν.

2.5 Automata and games

An *automaton over infinite words* 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 α is an acceptance condition. When $\mathcal{A}$ is a *generalized Büchi* or a *generalized co-Büchi* automaton, then $\alpha \subseteq 2^Q$ is a set of sets of accepting states. When $\mathcal{A}$ is a *parity* automaton, then $\alpha = \langle F_1, \ldots, F_d \rangle$, where the sets in α form a partition of Q. The number of sets in α is the *index* of $\mathcal{A}$. An automaton is *deterministic* if $|Q_0| = 1$ and for every $q \in Q$ and $\sigma \in \Sigma$, we have that $|\delta(q, \sigma)| = 1$. A run $r = r_0, r_1, \ldots$ 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 $i \geqslant 0$, 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$(r) = \{q : r_i = q$ for infinitely many $i \in \mathbb{N}\}$. The run r is *accepting* if it satisfies α. For generalized Büchi automata, a run is accepting if it visits all the sets in α infinitely often. Formally, for every set $F \in \alpha$, we have that inf$(r) \cap F \neq \emptyset$. Dually, in generalized co-Büchi automata, there should exist a set $F \in \alpha$ for which inf$(r) \cap F = \emptyset$. For parity automata, a run r is accepting if the minimal index i for which inf$(r) \cap F_i \neq \emptyset$ is even.

When $\mathcal{A}$ is a *nondeterministic* automaton, it accepts a word w if it has an accepting run of on w. When $\mathcal{A}$ is a *universal* automaton, it accepts a word w if all its runs on w are accepting. The language of $\mathcal{A}$, denoted $L(\mathcal{A})$, is the set of words that $\mathcal{A}$ accepts.

A *parity game* is $\mathcal{G} = \langle \Sigma_1, \Sigma_2, S, s_0, \delta, \alpha \rangle$, where Σ_1 and Σ_2 are alphabets for Players 1 and 2, respectively, S is a finite set of states, $s_0 \in S$ is an initial state, $\delta : S \times \Sigma_1 \times \Sigma_2 \to S$ is a transition function, and $\alpha = \langle F_1, \ldots, F_d \rangle$ is a parity acceptance condition, as described above. A play of the game starts in s_0. In each turn Player 1 chooses a letter $\sigma \in \Sigma_1$ and Player 2 chooses a letter $\tau \in \Sigma_2$. The play then moves from the current state s to the state $\delta(s, \sigma, \tau)$. Formally, a *play* of $\mathcal{G}$ is an infinite sequence $\rho = \langle s_0, \sigma_0, \tau_0 \rangle, \langle s_1, \sigma_1, \tau_1 \rangle, \ldots$ such that for every $i \geqslant 0$, we have that $s_{i+1} = \delta(s_i, \sigma_i, \tau_i)$. We define $\inf(\rho) = \{s \in S : s = s_i$ for infinitely many $i \in \mathbb{N}\}$. A play ρ is *winning for Player 1* if the minimal index i for which $\inf(\rho) \cap F_i \neq \emptyset$ is even. A *strategy* for Player 1 is a function $f : (S \times \Sigma_1 \times \Sigma_2)^* \times S \to \Sigma_1$ that assigns, for every finite prefix of a play, the next move for Player 1. Similarly, a strategy for Player 2 is a function $g : (S \times \Sigma_1 \times \Sigma_2)^* \times S \times \Sigma_1 \to \Sigma_2$. A strategy is *memoryless* if it does not depend on the history of the play. Thus, a memoryless strategy for Player 1 is a function $f : S \to \Sigma_1$ and for Player 2 it is a function $g : S \times \Sigma_1 \to \Sigma_2$.

A pair of strategies f, g for Players 1 and 2, respectively, induces a single play that conforms with the strategies. We say that Player 1 wins $\mathcal{G}$ if there exists a strategy f for Player 1 such that for every strategy g for Player 2, the play induced by f and g is winning for Player 1. Otherwise, Player 2 wins. By determinancy of Parity games [22], Player 2 wins $\mathcal{G}$ if there exists a strategy g for Player 2 such that for every strategy f of Player 1, the play induced by f and g is not winning for Player 1.

2.6 Solving the Boolean synthesis problem

The classical solution for the synthesis problem for LTL goes via games [25].[1] It involves a translation of the specification into a deterministic parity word automaton (DPW) over the alphabet $2^{I \cup O}$, which is then transformed into a game in which the players alphabets are 2^I and 2^O. More recent solutions avoids the determination and the solution of parity games and use instead alternating tree automata [20, 13]. The complexity of both approaches coincide. Below we describe the classical solution for the synthesis problem, along with its complexity, when the starting point is a specification given by a DPW.[2] In Remark 5.4, we describe an alternative, Safraless, approach, where the starting point is a universal co-Büchi automaton.

Theorem 2.2 *Consider a specification φ over I and O given by means of a DPW $\mathcal{D}_\varphi$ of size t over the alphabet $2^{I \cup O}$, with index k. The synthesis problem for φ can be solved in time $O(t^k)$.*

Proof: Let $\mathcal{D}_\varphi = \langle 2^{I \cup O}, Q, q_0, \delta, \alpha \rangle$. We define a game $\mathcal{G}_\varphi$ that models an interaction that simulates $\mathcal{D}_\varphi$ between a system (Player 1) that generates assignments in

[1] In [25] and other early works the games are formulated by means of tree automata.

[2] State-of-the-art algorithms for solving parity games achieve a better complexity [15, 29]. The bound, however, remains polynomial in the size of the game and exponential in its index. Since the challenge of solving parity games is orthogonal to our contribution here, we keep this component of our contribution simple.

2^O and an environment (Player 2) that generates assignments in 2^I. Formally, $\mathcal{G}_\varphi = \langle 2^O, 2^I, Q, q_0, \eta, \alpha \rangle$, where $\eta : Q \times 2^O \times 2^I \to Q$ is such that for every $q \in Q$, $i \in 2^I$, and $o \in 2^O$, we have that $\eta(q, i, o) = \delta(q, i \cup o)$. By [11], the game is determined and one of the players has a memoryless winning strategy. Such a strategy for Player 1 in $\mathcal{G}_\varphi$ is then a transducer that realizes φ. The game $\mathcal{G}_\varphi$ is of size $O(t)$ and index k. Hence, by [15, 29], we can find a memoryless strategy for the winner in time $O(t^k)$. □ □

3 Properties of LLTL

In this section we study properties of the logic LLTL. We focus on the set of attainable satisfaction values of an LLTL formula and on stability properties, namely the affect of perturbing the values of the atomic propositions on the satisfaction value of formulas.

3.1 Attainable values

Consider a lattice $\mathcal{L}$. We say that $\mathcal{L}$ is *pointed* if for all LLTL formulas φ, partitions $I \cup O$ of AP, and values $\ell_1, \ell_2 \in \mathcal{L}$, if φ is (I/O)-realizable with value ℓ_1 and with value ℓ_2, then φ is also (I/O)-realizable with value $\ell_1 \vee \ell_2$. Observe that if $\mathcal{L}$ is pointed, then every LLTL formula over $\mathcal{L}$ has a transducer that realizes it with a maximal value.

We start by showing that in general, not all lattices are pointed. In fact, our example has $O = \emptyset$, where (I/O)-realizability coincides with satisfiability. We then show that the lattices we focus on, are, however, pointed.

Theorem 3.1 *Not all distributive De-Morgan lattices are pointed.*

Proof: Consider the lattice $\mathcal{L} = \langle 2^{\{a,b\}} \times \{0,1\}, \leqslant \rangle$ where $\langle S_1, v_1 \rangle \leqslant \langle S_2, v_2 \rangle$ iff $v_1 \leqslant v_2$ or ($v_1 = v_2$ and $S_1 \subseteq S_2$). (See Figure 1). We define $\neg \langle S, v \rangle = \langle \{a, b\} \backslash S, 1 - v \rangle$. That is, negation negates both components. It is easy to verify that $\mathcal{L}$ is a distributive De-Morgan lattice.

Let $I = \{p\}$ and consider the formula $\varphi = (p \wedge \langle \{a\}, 1 \rangle) \vee (\neg p \wedge \langle \{b\}, 1 \rangle)$. Both $\langle \{a\}, 1 \rangle$ and $\langle \{b\}, 1 \rangle$ are attainable satisfaction values of φ. For example, by setting p to $\langle \{a\}, 1 \rangle$ or to $\langle \{a\}, 0 \rangle$. On the other hand, for every assignment ℓ to p, the second component of either ℓ or $\neg \ell$ is 0. Consequently, $\langle \{a, b\}, 1 \rangle$ is not attainable, thus $\mathcal{L}$ is not pointed. □ □

Theorem 3.2 *Fully-ordered lattices and power-set lattices are pointed.*

Proof: For fully-ordered lattices, we have $\ell_1 \vee \ell_2 \in \{\ell_1, \ell_2\}$, so pointed-ness is obvious. We prove the claim for power-set lattices. Consider a lattice $\mathcal{L} = \langle 2^X, \subseteq \rangle$ for some finite set X, and consider an LLTL formula φ over the atomic propositions $I \cup O$. For every

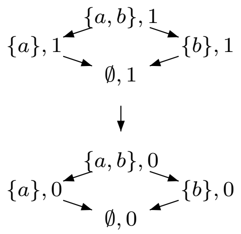


Figure 1: The lattice $\langle 2^{\{a,b\}} \times \{0,1\}, \leqslant\rangle$.

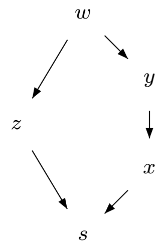


Figure 2: An N5 structure.

set $\ell \in \mathcal{L}$ and element $x \in X$, we define the *projection* $\ell_{|x}$ *of* ℓ *on* x to be `True` if $x \in \ell$ and `False` otherwise. We extend the definition of projection to a letter $\sigma \in \mathcal{L}^{I\cup O}$ by letting $p \in \sigma_{|x}$ iff $\mathcal{L}(p)_{|x} = \texttt{True}$. Thus, $\sigma_{|x} \subseteq I \cup O$. Finally, we extend the definition to a computation $\pi \in (\mathcal{L}^{I\cup O})^\omega$ by setting $(\pi_{|x})_i = (\pi_i)_{|x}$. Observe that $\pi_{|x} \in (2^{I\cup O})^\omega$.

For an element $x \in X$, let $\varphi_{|x}$ be the LTL formula obtained from φ by replacing every element $\ell \in \mathcal{L}$ that appears in φ by $\ell_{|x}$. Since the syntax of LLTL differs from that of LTL only by allowing elements from the lattice, it follows that $\varphi_{|x}$ is indeed an LTL formula.

We prove that for every computation $\pi \in (\mathcal{L}^{I\cup O})^\omega$ and for every $\ell \in \mathcal{L}$, it holds that $[\![\pi,\varphi]\!] \geqslant \ell$ iff $\pi_{|x} \models \varphi_x$ for all $x \in \ell$. Observe that $[\![\pi,\varphi]\!] \geqslant \ell$ iff $x \in [\![\pi,\varphi]\!]$ for all $x \in \ell$. From here the claim easily follows by induction on the structure of φ.

Now, assume that φ is (I/O)-realizable with value at least ℓ_1 and with value at least ℓ_2. We claim that φ is realizable with value $\ell_1 \vee \ell_2$. W.l.o.g we can assume $\ell_1 \cap \ell_2 = \emptyset$ (otherwise we replace ℓ_2 by $\ell_2 \backslash \ell_1$). Let $\mathcal{T}_1 = \langle \mathcal{L}, I, O, S^1, s_0^1, \eta^1, \tau^1\rangle$ and $T_2 = \langle \mathcal{L}, I, O, S^2, s_0^2, \eta^2, \tau^2\rangle$ be transducers that realize φ with values ℓ_1 and ℓ_2, respectively. We obtain from $\mathcal{T}_1$ and $\mathcal{T}_2$ a new transducer $\mathcal{T} = \langle \mathcal{L}, I, O, S^1 \times S^2, \langle s_0^1, s_0^2\rangle, \eta, \tau\rangle$ as follows. For every state $\langle s,t\rangle \in S^1 \times S^2$ and $\sigma \in \mathcal{L}^I$, we have $\eta(\langle s,t\rangle, \sigma) = \langle \eta^1(s,\sigma), \eta^2(t,\sigma)\rangle$. For every state $\langle s,t\rangle \in S^1 \times S^2$ and for every $o \in O$, we have $\tau(\langle s,t\rangle)(o) = (\ell_1 \cap \tau^1(s)(o)) \cup (\ell_2 \cap \tau^2(s)(o))$. We claim that $\mathcal{T}$ realizes φ with value $\ell_1 \vee \ell_2$.

Consider an environment-computation $\pi \in (\mathcal{L}^I)^\omega$, and consider the corresponding computations $\rho, \rho' \in \mathcal{L}^{I\cup O}$ of $\mathcal{T}_1$ and $\mathcal{T}_2$, respectively. It holds that $[\![\rho,\varphi]\!] \geqslant \ell_1$ and $[\![\rho',\varphi]\!] \geqslant \ell_2$.

Consider the output computation $\theta \in (\mathcal{L}^O)^\omega$ of $\mathcal{T}$ on the input π. By the construction of $\mathcal{T}$, it is easy to prove that $\theta_i(o) = (\ell_1 \cap \rho_i(o)) \cup (\ell_2 \cap \rho_i'(o))$.

Consider the computation π' of $\mathcal{T}$ obtained by combining π and θ. For every $x \in \ell_1$, we have that $\pi'_{|x} = \rho_{|x}$. Thus, $\pi'_{|x} \models \varphi_x$ for every $x \in \ell_1$. Similarly, for every $x \in \ell_2$, we have that $\pi'_{|x} = \rho'_{|x}$, so $\pi'_{|x} \models \varphi_x$ for every $x \in \ell_2$. We conclude that for every $x \in \ell_1 \cup \ell_2$ it holds that $\pi'_{|x} \models \varphi$, and so $[\![\pi',\varphi]\!] \geqslant \ell_1 \vee \ell_2$. Thus, $\mathcal{T}$ realizes φ with value $\ell_1 \vee \ell_2$. □□

3.2 Stability

For two computations $\pi = \pi_0, \pi_1, \ldots$ and $\pi' = \pi'_0, \pi'_1, \ldots$, both in $(\mathcal{L}^{AP})^\omega$, we define the *global distance* between π and π', denoted $gd(\pi, \pi')$, as $\sum_{i \geqslant 0} d(\pi_i, \pi'_i)$. Note that $gd(\pi, \pi')$ may be infinite. We define the *local distance* between π and π', denoted $ld(\pi, \pi')$, as $\max_{i \geqslant 0} d(\pi_i, \pi'_i)$. Note that $ld(\pi, \pi') \leqslant |\mathcal{L}|$.

Consider an LLTL formula φ over AP and $\mathcal{L}$. We say that φ is *globally stable* if for every pair π and π' of computations, we have $d([\![\pi, \varphi]\!], [\![\pi', \varphi]\!]) \leqslant gd(\pi, \pi')$. Thus, the difference between the satisfaction value of φ in π and π' is bounded by the sum of differences between matching locations in π and π'. Also, φ is *locally stable* if for every pair π and π' of computations, we have $d([\![\pi, \varphi]\!], [\![\pi', \varphi]\!]) \leqslant ld(\pi, \pi')$. Thus, the difference between the satisfaction value of φ in π and π' is bounded by the maximal difference between matching locations in π and π'. Here, we study stability of all LLTL formulas. In Section 5.3, we study the problem of deciding whether a given LLTL formula is stable, and discuss the relevancy of stability to synthesis with noise.

Consider an LLTL formula φ over the atomic propositions AP, and consider computations $\pi, \pi' \in (\mathcal{L}^{AP})^\omega$. Assume that $gd(\pi, \pi') \leqslant 1$. That is, π and π' differ only in one location, where they differ in the value of a single atomic proposition, whose value in π is a child of its value in π' or vice versa. It is tempting to think that then, $d([\![\pi, \varphi]\!], [\![\pi', \varphi]\!]) \leqslant 1$, which would imply that φ should be globally stable.

We start by breaking this intuition, showing that for non-distributive lattices, this is false. The proof makes use of an *N5 structure*, depicted in Figure 2. Formally, an N5 structure in a lattice $\mathcal{L}$ is a tuple $\langle x, y, z, w, s \rangle$ such that the following relations hold: $s < x < y < w$, $s < z < w$, $y \nleqslant z$, $z \nleqslant y$, $x \nleqslant z$, and $z \nleqslant x$. Note that $x \vee (z \wedge y) = x \vee s = x$, whereas $(x \vee z) \wedge (x \vee y) = w \wedge y = y$. Hence, the structure of N5 is never a sub-lattice in a distributive lattice.

Theorem 3.3 LLTL *formulas may not be globally stable with respect to non-distributive lattices.*

Proof: Consider the lattice N5, the formula $\varphi = p \vee q$, and a computation π such that $\pi_0(p) = s$ and $\pi_0(q) = x$. Clearly $[\![\pi, \varphi]\!] = x$. Now, let π' be the computation obtained from π by setting $\pi'_0(p) = z$. It holds that $gd(\pi, \pi') = 1$. However, $[\![\pi', \varphi]\!] = z \vee x = w$, and $d(x, w) = 2$. Thus, φ is not globally stable over the lattice N5. □ □

We now proceed to show that when defined with respect to a distributive lattice, all LLTL formulas are globally stable.

Theorem 3.4 LLTL *formulas are globally stable with respect to De-Morgan distributive lattices.*

Proof: We prove that for every LLTL formula φ and computations $\pi, \pi' \in (\mathcal{L}^{AP})^\omega$, if $gd(\pi, \pi') = 1$, then $d([\![\pi, \varphi]\!], [\![\pi', \varphi]\!]) \leqslant 1$. The result then follows by induction on $gd(\pi, \pi')$.

Consider an LLTL formula φ and computations π, π' such that $gd(\pi, \pi') = 1$. That is, there exists a single index $i \geqslant 0$ such that $d(\pi_i, \pi'_i) = 1$ and $\pi_j = \pi'_j$ for all $j \neq i$. W.l.o.g, there is $p \in AP$ such that $\pi_i(p) \preceq \pi'_i(p)$. By Birkhoff's representation theorem, there exists a unique element $u \in \mathrm{JI}(\mathcal{L})$ such that $\pi'_i(p) = \pi_i(p) \vee u$. We prove, by induction over the structure of φ, that $[\![\pi', \varphi]\!] \in \{[\![\pi, \varphi]\!] \wedge \neg u, [\![\pi, \varphi]\!], [\![\pi, \varphi]\!] \vee u\}$ and that $d([\![\pi', \varphi]\!], [\![\pi, \varphi]\!]) \leqslant 1$.

To simplify the proof, we observe that since π and π' differ only in a single index, and in particular only in a finite prefix, we can avoid treating the case of U subformulas. Indeed, we can expand subformulas of the form $\psi \mathsf{U} \theta$ using propositional conjunctives and the X operator so that all the suffixes before π^i are not evaluated on subformulas that contain U. Clearly, the value of the remaining evaluation of U subformulas does not change, as π and π' agree on suffixes that start after the i-th position.

- If $\varphi = \ell \in \mathcal{L}$ the claim is trivial.

- If $\varphi = p \in AP$, then $d([\![\pi, \varphi]\!], [\![\pi', \varphi]\!]) = d([\![\pi_0, p]\!], [\![\pi'_0, p]\!])$, which, by the assumption, is at most 1. Moreover, $[\![\pi', p]\!] \in \{[\![\pi, p]\!], [\![\pi, p]\!] \vee u\}$.

- If $\varphi = \neg\psi$, then by the induction hypothesis it holds that $[\![\pi', \psi]\!] \in \{[\![\pi, \psi]\!] \wedge \neg u, [\![\pi, \psi]\!], [\![\pi, \psi]\!] \vee u\}$ and $d([\![\pi, \psi]\!], [\![\pi', \psi]\!]) \leqslant 1$. Hence, by the properties of negation, $d(\neg[\![\pi, \psi]\!], \neg[\![\pi', \psi]\!]) \leqslant 1$ and $[\![\pi', \varphi]\!] \in \{[\![\pi, \varphi]\!] \wedge \neg u, [\![\pi, \varphi]\!], [\![\pi, \varphi]\!] \vee u\}$, and we are done.

- If $\varphi = \psi \vee \theta$, then, by the induction hypothesis, it holds in particular that $[\![\pi, \psi]\!] \wedge \neg u \leqslant [\![\pi', \psi]\!] \leqslant [\![\pi, \psi]\!] \vee u$ and $[\![\pi, \theta]\!] \wedge \neg u \leqslant [\![\pi', \theta]\!] \leqslant [\![\pi, \theta]\!] \vee u$. Therefore, $[\![\pi, \varphi]\!] \wedge \neg u \leqslant [\![\pi', \varphi]\!] \leqslant [\![\pi, \varphi]\!] \vee u$. Figure 3 demonstrates the above relations. Intuitively, adding a lattice element between $\psi \vee \theta \vee u$ and $(\psi \vee \theta) \wedge u$ induces an N5-structure. Thus, $[\![\pi', \varphi]\!]$ must be within distance 1 of $[\![\pi, \varphi]\!]$. This is the key point in the proof.

  Formally, in order to prove that the distance is preserved, we distinguish between cases. First, if $[\![\pi', \varphi]\!] = [\![\pi, \varphi]\!]$, then we are done. If $[\![\pi', \varphi]\!] = [\![\pi, \varphi]\!] \vee v$, then since $[\![\pi', \varphi]\!] = [\![\pi', \psi]\!] \vee [\![\pi', \theta]\!]$, it must be that w.l.o.g $[\![\pi', \psi]\!] = [\![\pi, \psi]\!] \vee u$. That is, at least one of ψ and θ gets joined with u. Assume by way of contradiction that there exists $t \in \mathcal{L}$ such that $[\![\pi', \varphi]\!] < t < [\![\pi, \varphi]\!] \vee v$. We then have the N5 structure[3] $\langle \psi \vee \theta, t, \psi \vee u, \psi, \psi \vee \theta \vee u \rangle$. Since, however, $\mathcal{L}$ is distributive, it cannot have an N5 structure, and we have reached a contradiction.

[3] It may be the case that some of the nodes coincide, and this is not a proper N5. However, these cases are easy to handle.

The case $[\![\pi,\varphi]\!] \wedge \neg u \leqslant [\![\pi',\varphi]\!]$ is handled similarly.

$$\begin{tikzcd}
 & \psi \vee \theta \vee u \arrow[dl] \arrow[d] \arrow[dr] & \\
\psi \vee u \arrow[d] & \psi \vee \theta \arrow[dl] \arrow[d] \arrow[dr] & \theta \vee u \arrow[d] \\
\psi \arrow[d] & (\psi \vee \theta) \wedge \neg u \arrow[dl] \arrow[dr] & \theta \arrow[d] \\
\psi \wedge \neg u & & \theta \wedge \neg u
\end{tikzcd}$$

Figure 3: The relations described in the disjunction case in the proof of Theorem 3.4.

- If $\varphi = \mathsf{X}\psi$, the claim follows immediately from the induction hypothesis.

□ □

We now turn to study local stability. Since local stability refers to the maximal change along a computation, it is a very permissive notion. In particular, it is not hard to see that in a fully-ordered lattice, a local change of 1 entails a change of at most 1 in the satisfaction value. Thus, we have the following.

Theorem 3.5 LLTL *formulas are locally stable with respect to fully-ordered lattices.*

In partially-ordered lattices, however, things are more involved, as local changes may be in different "directions". Formally, we have the following.

Theorem 3.6 LLTL *formulas may not be locally stable.*

Proof: Consider the power-set lattice $\langle 2^{\{a,b\}}, \subseteq \rangle$ and the LLTL formula $\varphi = p \vee \mathsf{X}p$. Consider computations π and π' with $\pi_0(p) = \pi_1(p) = \emptyset$, $\pi'_0(p) = \{a\}$, and $\pi'_1(p) = \{b\}$. It holds that $ld(\pi,\pi') = 1$, whereas $d([\![\pi,\varphi]\!],[\![\pi',\varphi]\!]) = d(\emptyset,\{a,b\}) = 2$. We conclude that φ is not locally stable. □ □

4 Translating LLTL to Automata

In this section we describe an automata-theoretic approach for reasoning about LLTL specifications. One approach is to develop a framework that is based on *lattice automata* [17]. Like LLTL formulas, lattice automata map words to values in a lattice. Lattice automata have proven to be useful in solving the satisfiability and the model-checking problems for LLTL [17]. However, the solution of the synthesis problem involves automata-theoretic constructions for which the latticed counterpart is either not known or is very complicated. In particular, Safra's determinization construction has not yet been studied for lattice automata, and a latticed counterpart of it is not going to be of much fun. Likewise, the solution of two-player games (even reachability, and moreover parity) in the latticed setting is much more complicated than in the Boolean setting. In particular, obtaining a value $\ell_1 \vee \ell_2$ in a latticed game may require one strategy for obtaining ℓ_1 and a different strategy for obtaining ℓ_2 [18]. When the game is induced by a realizability problem, it is not clear how to combine such strategies into a single transducer that realizes the underlying specification with value $\ell_1 \vee \ell_2$.

Accordingly, a second approach, which is the one we follow, is to use Boolean automata. The fact that LLTL formulas have finitely many possible satisfaction values suggests that this is possible. For fully-ordered lattices, a similar approach has been taken in [12, 1]. Beyond the challenge in these works of maintaining the simplicity of the automata-theoretic framework of LTL, an extra challenge in the latticed setting is caused by the fact values may be only partially ordered. We will elaborate on this point below.

In order to explain our framework, let us recall first the translation of LTL formulas to nondeterministic generalized Büchi automata (NGBW), as introduced in [31]. There, each state of the automaton is associated with a set of formulas, and the NGBW accepts a computation from a state q iff the computation satisfies exactly all the formulas associated with q. The state space of the NGBW 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 the construction here, each state of the NGBW assigns a satisfaction value to every subformula. While it is not difficult to extend the local consistency rules to the latticed settings, handling of eventualities is more complicated. To see why, consider for example the formula Fp, for $p \in AP$, and the computation π in which the satisfaction value of p is $(\{a\}, \{b\}, \{c\})^\omega$. While $[\![\pi, Fp]\!] = \{a, b, c\}$, the computation never reaches a position in which the satisfaction value of the eventuality p is $\{a, b, c\}$. This poses a problem on translations of LTL formulas to automata, where eventualities are handed by making sure that each state in which the satisfaction of $\psi_1 \mathsf{U} \psi_2$ is guaranteed, is followed by a state in which the satisfaction of ψ_2 is guaranteed. For a multi-valued setting with fully-ordered values, as is the case in [12, 1], the latter can be replaced by a requirement to visit a state in which the guaranteed satisfaction value of ψ exceeds that of $\psi_1 \mathsf{U} \psi_2$. As the example above demonstrates, such a position need not exist when the values are partially ordered. In order to address the above problem, every state in the NGBW associates with every subformula of the form $\psi_1 \mathsf{U} \psi_2$ a value in $\mathcal{L}$ that ψ_2 still needs "accumulate" in order for $\psi_1 \mathsf{U} \psi_2$ to have its assigned satisfaction value. Thus, as in other break-point constructions [31, 23], we decompose the requirement to obtain a value ℓ to requirements to obtain join-irreducible values whose join is ℓ, and we check these requirements together.

Theorem 4.1 *Let φ be an* LLTL *formula over $\mathcal{L}$ and $P \subseteq \mathcal{L}$ be a predicate. There exists an NGBW $\mathcal{A}_{\varphi,P}$ such that for every computation $\pi \in (2^{AP})^{\omega}$, it holds that $[\![\pi,\varphi]\!] \in P$ iff $\mathcal{A}_{\varphi,P}$ accepts π. The state space and transitions of $\mathcal{A}_{\varphi,P}$ are independent of P, which only influences the set of initial states. The NGBW $\mathcal{A}_{\varphi,P}$ has at most $|\mathcal{L}|^{O(|\varphi|)}$ states and index at most $|\varphi|$.*

Proof: We define $\mathcal{A}_{\varphi,P} = \langle \mathcal{L}^{AP}, Q, \delta, Q_0, \alpha \rangle$ as follows. Let $cl(\varphi)$ be the set of φ's subformulas, and let $ucl(\varphi)$ be the set of φ's subformulas of the form $\psi_1 \mathsf{U} \psi_2$. Let G_φ and F_φ be the collection of functions $g : cl(\varphi) \to \mathcal{L}$ and $f : ucl(\varphi) \to \mathcal{L}$, respectively. For an element $v \in \mathcal{L}$, let $\mathrm{JI}(v)$ be the minimal set $S \subseteq \mathrm{JI}(\mathcal{L})$ such that $v = \bigvee_{s \in S} s$. By Birkhoff's theorem, this set is well defined, and the JI mapping is a bijection.

For a pair of functions $\langle g, f \rangle \in G_\varphi \times F_\varphi$, we say that $\langle g, f \rangle$ is *consistent* if for every $\psi \in cl(\varphi)$, the following holds.

- If $\psi = v \in \mathcal{L}$, then $g(\psi) = v$.
- If $\psi = \neg\psi_1$, then $g(\psi) = \neg g(\psi_1)$.
- If $\psi = \psi_1 \vee \psi_2$, then $g(\psi) = g(\psi_1) \vee g(\psi_2)$.
- If $\psi = \psi_1 \mathsf{U} \psi_2$, then $\mathrm{JI}(f(\psi)) \cap \mathrm{JI}(g(\psi_2)) = \emptyset$.

The state space Q of $\mathcal{A}_{\varphi,\ell}$ is the set of all consistent pairs of functions in $G_\varphi \times F_\varphi$. Intuitively, while the function g describes the satisfaction value of the formulas in the closure, the function f describes, for each subformula of the form $\psi_1 \mathsf{U} \psi_2$, the values in which ψ_2 still has to be satisfied in order for the satisfaction value $g(\psi_1 \mathsf{U} \psi_2)$ to be fulfilled. Accordingly, if a value is in $\mathrm{JI}(g(\psi_2))$, it can be removed from $f(\psi_1 \mathsf{U} \psi_2)$, explaining why $\mathrm{JI}(f(\psi_1 \mathsf{U} \psi_2)) \cap \mathrm{JI}(g(\psi_2)) = \emptyset$.

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 two states $\langle g, f \rangle$ and $\langle g', f' \rangle$ in Q and a letter $\sigma \in \mathcal{L}^{AP}$, we have that $\langle g', f' \rangle \in \delta(\langle g, f \rangle, \sigma)$ iff the following hold.

- For all $p \in AP$, we have that $\sigma(p) = g(p)$.
- For all $\mathsf{X}\psi_1 \in cl(\varphi)$, we have $g(\mathsf{X}\psi_1) = g'(\psi_1)$.
- For all $\psi_1 \mathsf{U} \psi_2 \in cl(\varphi)$, we have $g(\psi_1 \mathsf{U} \psi_2) = g(\psi_2) \vee (g(\psi_1) \wedge g'(\psi_1 \mathsf{U} \psi_2))$ and

$$f'(\psi_1 \mathsf{U} \psi_2) = \begin{cases} \mathrm{JI}(f(\psi_1 \mathsf{U} \psi_2)) \backslash \mathrm{JI}(g'(\psi_2)) & \text{If } \mathrm{JI}(f(\psi_1 \mathsf{U} \psi_2)) \neq \emptyset, \\ \mathrm{JI}(g'(\psi_1 \mathsf{U} \psi_2)) \backslash \mathrm{JI}(g'(\psi_2)) & \text{Otherwise.} \end{cases}$$

Finally, every formula of the form $\psi_1 \mathsf{U} \psi_2$ contributes to the acceptance condition α the set $F_{\psi_1 \mathsf{U} \psi_2} = \{\langle g, f\rangle : \ \mathrm{JI}(f(\psi_1 \mathsf{U} \psi_2)) = \varnothing\}$.

Observe that while δ is nondeterministic, it is only nondeterministic in the first component. That is, once the function g' is chosen, there is a single function f' that can match the transition.

We now proceed to prove the correctness of the construction and analyze the blow-up it involves. In the proof, we identify a set $S \subseteq \mathrm{JI}(\mathcal{L})$ with the element $\bigvee_{s \in S} s \in \mathcal{L}$. Observe that it suffices to prove that for every $\ell \in \mathcal{L}$, the NGBW $\mathcal{A}_{\varphi,\{\ell\}}$ accepts a computation π iff $[\![\pi, \varphi]\!] = \ell$. We first prove that if $\pi \in (\mathcal{L}^{AP})^\omega$ is such that $[\![\pi, \varphi]\!] = \ell$ for some $\ell \in \mathcal{L}$, then $\mathcal{A}_{\varphi,\{\ell\}}$ accepts π. For every $i \in \mathbb{N}$, let $g_i \in G_\varphi$ be such that for all $\psi \in cl(\varphi)$, we have that $g_i(\psi) = [\![\pi^i, \psi]\!]$. Also, let $f_0 : ucl(\varphi) \to \mathcal{L}$ be such that for every subformula of the form $\psi_1 \mathsf{U} \psi_2$, we have $f_0(\psi_1 \mathsf{U} \psi_2) = \mathrm{JI}(g_0(\psi_1 \mathsf{U} \psi_2)) \backslash \mathrm{JI}(g_0(\psi_2))$. Finally, for $i \in \mathbb{N}$, let f_{i+1} be induced from f_i and g_{i+1} in the single way that satisfies the conditions in the definition of δ.

We claim that $r = \langle g_0, f_0\rangle, \langle g_1, f_1\rangle, \ldots$ is an accepting run of $\mathcal{A}_{\varphi,\{l\}}$ on π. First, the semantics of LLTL implies that the consistency conditions, both the local ones and these imposed by δ are satisfied. In particular, for the conditions imposed by δ, this follows from the fact that for all positions $i \in \mathbb{N}$, we have that $[\![\pi^i, X\psi_1]\!] = [\![\pi^{i+1}, \psi_1]\!]$ and $[\![\pi^i, \psi_1 \mathsf{U} \psi_2]\!] = [\![\psi_2]\!] \vee ([\![\pi^i, \psi_1]\!] \wedge [\![\pi^i, \psi_1 \mathsf{U} \psi_2]\!])$. Also, since $g_0(\varphi) = \ell$, then $\langle g_0, f_0\rangle \in Q_0$

It is left to prove that r is accepting. Consider a sub-formula of the form $\psi_1 \mathsf{U} \psi_2$. We prove that r visits $F_{\psi_1 \mathsf{U} \psi_2}$ infinitely often. Consider a position $i \in \mathbb{N}$ and let $[\![\pi^i, \psi_1 \mathsf{U} \psi_2]\!] = y$. We prove that there is a position $n \geqslant i$ such that $f_n = \varnothing$, thus $\langle g_n, f_n\rangle \in F_{\psi_1 \mathsf{U} \psi_2}$. By the semantics of U and the finiteness of $\mathcal{L}$, there is a (minimal) index $n \geqslant i$ such that $y = \bigvee_{i \leqslant j \leqslant n}([\![\pi^j, \psi_2]\!] \wedge \bigwedge_{i \leqslant k < j}[\![\pi^k, \psi_1]\!])$. It is easy to prove by induction on $n - i$ that there exists some $i \leqslant k \leqslant n$ such that $\mathrm{JI}(f_k(\psi_1 \mathsf{U} \psi_2)) = \varnothing$, using the fact that $f_j(\psi_1 \mathsf{U} \psi_2) \leqslant g_j(\psi_1 \mathsf{U} \psi_2)$ for all $j \geqslant 0$.

The other direction is more complicated. Let $\pi \in (\mathcal{L}^{AP})^\omega$ be such that π is accepted by $\mathcal{A}_{\varphi,\{\ell\}}$. We prove that $[\![\pi, \varphi]\!] = \ell$. Let $\rho = \langle g_1, f_1\rangle, \langle g_2, f_2\rangle, \ldots$ be an accepting run of $\mathcal{A}_{\varphi,\{\ell\}}$ on π, and let $h_1, h_2, \ldots \in (G_\varphi)^\omega$ be 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 $\psi \in cl(\varphi)$. If $\psi = p \in AP$, then since ρ is a legal run, a transition from state $\langle g_i, f_i\rangle$ is possible with letter σ iff $\sigma(p) = [\![\pi^i, p]\!] = \rho(p)$, and we are done. If $\psi = v \in \mathcal{L}$, $\psi = \psi_1 \vee \psi_2$, or $\psi = \mathsf{X}\psi_1$, then the claim follows from the consistency rules and the induction hypothesis. Finally, if $\psi = \psi_1 \mathsf{U} \psi_2$, then, as we prove in Lemma 4.2 below, the fact that ρ is an accepting run implies the first equality in the chain below. The second equality follows from the induction hypothesis, and the third equality is from the semantics of LLTL.

$$g_i(\psi) = \bigvee_{i \leqslant j}(\, g_j(\psi_2) \wedge \bigwedge_{i \leqslant k < j} g_k(\psi_1)\,) = \bigvee_{i \leqslant j}(\, [\![\pi^j, \psi_2]\!] \wedge \bigwedge_{i \leqslant k < j} [\![\pi^k, \psi_1]\!]\,) = [\![\pi^i, \psi]\!].$$

We conclude that $h_0 = g_0$. Since $g_0 \in Q_0$, it follows that $[\![\pi, \varphi]\!] = \ell$ and we are done.

□ □

Lemma 4.2 $g_i(\psi_1 \mathsf{U} \psi_2) = \bigvee_{i \leqslant j}(g_j(\psi_2) \wedge \bigwedge_{i \leqslant k < j} g_k(\psi_1))$.

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

$$g_i(\psi_1 \mathsf{U} \psi_2) = g_i(\psi_2) \vee (g_i(\psi_1) \wedge g_{i+1}(\psi_1 \mathsf{U} \psi_2)). \quad (*)$$

We prove the lemma by proving that for every $v \in \mathrm{JI}(\mathcal{L})$ it holds that $g_i(\psi_1 \mathsf{U} \psi_2) \geqslant v$ iff $\bigvee_{i \leqslant j}(g_j(\psi_2) \wedge \bigwedge_{i \leqslant k < j} g_k(\psi_1) \geqslant v$. The equality then follows from Birkhoff's theorem.

Using $(*)$, it is easy to prove by induction that for every index i and for every $n \in \mathbb{N}$ it holds that

$$g(\psi_1 \mathsf{U} \psi_2) = \bigvee_{i \leqslant j \leqslant n} \left(g_j(\psi_2) \wedge \bigwedge_{i \leqslant k < j} g_k(\psi_1) \right) \vee \left(g_{n+1}(\psi_1 \mathsf{U} \psi_2) \wedge \bigwedge_{i \leqslant k \leqslant n} g_k(\psi_1) \right).$$

We denote the above equation by $(**)$.

Let $v \in \mathrm{JI}(\mathcal{L})$ and assume that

$$\bigvee_{i \leqslant j}(g_j(\psi_2) \wedge \bigwedge_{i \leqslant k < j} g_k(\psi_1)) \geqslant v.$$

Since v is join-irreducible, it follows that there exist some $n \in \mathbb{N}$ such that

$$g_n(\psi_2) \wedge \bigwedge_{i \leqslant k < n} g_k(\psi_1) \geqslant v.$$

Since $(**)$ is true for every n, then in particular, we have that $g(\psi_1 \mathsf{U} \psi_2) \geqslant v$, which concludes the first direction of the proof.

For the second direction, assume that $g(\psi_1 \mathsf{U} \psi_2) \geqslant v$. If there exists $n \geqslant i$ such that $\bigvee_{i \leqslant j \leqslant n} \left(g_j(\psi_2) \wedge \bigwedge_{i \leqslant k < j} g_k(\psi_1) \right) \geqslant v$, then we are done. Assume by way of contradiction that there is no such n. Thus, by $(**)$, for every $n \geqslant i$ it holds that $g_{n+1}(\psi_1 \mathsf{U} \psi_2) \wedge \bigwedge_{i \leqslant k \leqslant n} g_k(\psi_1) \geqslant v$, which means that $g_{n+1}(\psi_1 \mathsf{U} \psi_2) \geqslant v$ and $\bigwedge_{i \leqslant k \leqslant n} g_k(\psi_1) \geqslant v$. In particular, there cannot exist $n \geqslant i$ such that $g_n(\psi_2) \geqslant v$, otherwise it would contradict our assumption.

Since the run is accepting, there exists $n_1 > i$ such that $f_{n_1}(\psi_1 \mathsf{U} \psi_2) = \varnothing$. Consider the suffix of the run starting from $n_1 + 1$. For every $t \geqslant n_1$, We have that $g_t(\psi_1 \mathsf{U} \psi_2) \geqslant v$ but $g_t(\psi_2) \not\geqslant v$. Thus, $v \in \mathrm{JI}(f_{n+1}(\psi_1 \mathsf{U} \psi_2))$, and v will never be removed from the f component, this is in contradiction to the fact that the run is accepting, and we are done.

□ □

5 LLTL Synthesis

Recall that in the synthesis problem we are given an LLTL formula φ over sets I and O of input and output variables, taking truth values from a lattice $\mathcal{L}$, and we want to generate an (I/O)-transducer over $\mathcal{L}$ all whose computations satisfy φ in a value from some desired set P of satisfaction values. In the noisy setting, the transducer may read a perturbed value of the input signals, and still all its computations need to satisfy φ as required. In this section we use the construction in Theorem 4.1 in order to solve both variants of the synthesis problem.

5.1 Solving the LLTL synthesis problem

Theorem 5.1 *The synthesis problem for* LLTL *is 2EXPTIME-complete. Given an* LLTL *formula φ over a lattice $\mathcal{L}$ and a predicate $P \subseteq \mathcal{L}$, we can solve the synthesis problem for $\langle \varphi, P \rangle$ in time $2^{|\mathcal{L}|^{O(|\varphi|)}}$.*

Proof: Let m denote the size of $\mathcal{L}$, and let n denote the length of φ. The construction in Theorem 4.1 yields an NGBW with $m^{O(n)}$ states and index n. By determinizing the NGBW we obtain an equivalent DPW $\mathcal{D}_{\varphi,P}$ of size $2^{m^{O(n)} \log m^{O(n)}} = 2^{O(n)m^{O(n)}} = 2^{m^{O(n)}}$ and index $m^{O(n)}$ [28, 24]. Following the same lines as the proof of Theorem 2.2, we see that in order to solve the LLTL synthesis problem, it suffices to solve the parity game that is obtained from $\mathcal{D}_{\varphi}$, except that here the alphabets of Players 1 and 2 are $\mathcal{L}^O$ and $\mathcal{L}^I$, respectively. Accordingly, a winning memoryless strategy for Player 1 is an (I/O)-transducer over $\mathcal{L}$ that realizes $\langle \varphi, P \rangle$.

As stated in Theorem 2.2, the parity game that is obtained from $\mathcal{D}_{\varphi,P}$ can be solved in time $(2^{m^{O(n)}})^{m^{O(n)}} = 2^{m^{O(n)}}$. We conclude that the LLTL-synthesis problem is in 2EXPTIME. Hardness in 2EXPTIME follow from the hardness of the synthesis problem in the Boolean setting, which corresponds to a fully-ordered lattice with two values. □□

5.2 Solving the noisy LLTL synthesis problem

Consider an LLTL formula φ over the atomic propositions $I \cup O$, a predicate $P \subseteq \mathcal{L}$, and a noise function $\nu : \mathcal{L}^I \to 2^{\mathcal{L}^I}$. Recall that the goal in noisy synthesis is to find a transducer $\mathcal{T}$ that realizes $\langle \varphi, P \rangle$ with noise ν. Our goal is to construct a DPW on which we can apply the algorithm described in Theorem 2.2. For this, we proceed in three steps. First, we translate φ to a universal generalized co-Büchi word automaton (UGCW). Then, we incorporate the noise in the constructed UGCW. Finally, we determinize the UGCW to obtain a DPW, from which we proceed as described in Theorem 2.2. We start by showing how to incorporate noise in universal automata.

Lemma 5.2 *Consider a UGCW $\mathcal{D}$ and a noise function ν. There exists a UGCW $\mathcal{D}'$ such that $\mathcal{D}'$ accepts a computation ρ iff $\mathcal{D}$ accepts every computation ρ' that is ν-indistinguishable from ρ. Moreover, $\mathcal{D}'$ has the same state space and acceptance condition as $\mathcal{D}$.*

Proof: Let $\mathcal{D} = \langle I \cup O, Q, Q_0, \delta, \alpha\rangle$. We obtain $D' = \langle I \cup O, Q, Q_0, \delta', \alpha\rangle$ from $\mathcal{D}$ by modifying δ as follows. For every $\sigma \in I \cup O$, let $\Gamma_\sigma = \{\gamma : \gamma|_O = \sigma|_O \text{ and } \gamma|_I \in \nu(\sigma|_I)\}$. Thus, Γ_σ contains all letters that are ν-indistinguishable from σ. Then, for every state $q \in Q$, we have that $\delta'(q, \sigma) = \bigcup_{\gamma \in \Gamma_\sigma} \delta(q, \gamma)$. Thus, reading the letter σ, the UGCW $\mathcal{D}'$ simulates all the runs of $\mathcal{D}$ on all the letters that $\mathcal{D}$ may read when the actual letter in the input is σ.

It is not hard to show that the set of runs of $\mathcal{D}'$ on a computation ρ is exactly the set of all the runs of $\mathcal{D}$ on all the computations that are ν-indistinguishable from ρ. From this, the correctness of the construction follows. □ □

Theorem 5.3 *The noisy synthesis problem for* LLTL *is 2EXPTIME-complete. Given an* LLTL *formula φ over a lattice $\mathcal{L}$, a predicate $P \subseteq \mathcal{L}$, and a noise function ν, we can solve the synthesis problem for $\langle\varphi, P\rangle$ with noise ν in time $2^{m^{O(n)}}$.*

Proof: Let $\overline{P} = \mathcal{L} \backslash P$, and let $\mathcal{A}_{\varphi,\overline{P}}$ be the NGBW constructed for φ and $\overline{P}$ in Theorem 4.1. Observe that $\mathcal{A}_{\varphi,\overline{P}}$ accepts a computation ρ iff $[\![\rho, \varphi]\!] \notin P$. Next, we dualize $\mathcal{A}_{\varphi,\overline{P}}$ and obtain a UGCW $\mathcal{D}_{\varphi,P}$ for the complement language, namely all computations ρ such that $[\![\rho, \varphi]\!] \in P$. We now apply the procedure in Lemma 5.2 to $\mathcal{D}_{\varphi,P}$ and obtain a UGCW $\mathcal{D}'_{\varphi,P}$ that accepts a computation ρ iff $\mathcal{D}_{\varphi,P}$ accepts every computation ρ' that is ν-indistinguishable from ρ. Next, we determinize $\mathcal{D}'_{\varphi,P}$ to an equivalent DPW $\mathcal{D}''_{\varphi,P}$.

We claim that the algorithm described in the proof of Theorem 2.2 can be applied to $\mathcal{D}''_{\varphi,P}$. To see this, let $\mathcal{D}''_{\varphi,P} = \langle I \cup O, S, s_0, \eta, \beta\rangle$ and consider the game $\mathcal{G}$ that is obtained from $\mathcal{D}''_{\varphi,P}$. That is, $\mathcal{G} = \langle \mathcal{L}^O, \mathcal{L}^I, S, s_0, \eta, \beta\rangle$, where for every $q \in S$, $i \in \mathcal{L}^I$, and $o \in \mathcal{L}^O$, we have that $\eta(q, i, o) = \mu(q, i \cup o)$.

A (memoryless) winning strategy f for Player 1 in $\mathcal{G}$ is then an (I/O)-transducer over $\mathcal{L}$ with the following property: for every strategy g of the environment, consider the play ρ that is induced by f and g. The play ρ induces a computation $w \in \mathcal{L}^{I \cup O}$ that is accepted by $\mathcal{D}''_{\varphi,P}$. By the construction of $\mathcal{D}''_{\varphi,P}$, this means that for every computation w' that is ν-indistinguishable from w, the run of $\mathcal{D}_{\varphi,P}$ on w' is accepting. Hence, $[\![w', \varphi]\!] \in P$, which in turn implies that f realizes $\langle\varphi, P\rangle$ with noise ν.

We now analyze the complexity of the algorithm. Let m denote the size of $\mathcal{L}$, and let n denote the length of φ. By Theorem 4.1, the size of $\mathcal{A}_{\varphi,\overline{P}}$ is $m^{O(n)}$ and it has index at most n. Dualizing results in a UGCW of the same size and acceptance condition, and so is the transition to $\mathcal{D}'_{\varphi,P}$. Determinization involves an exponential blowup, such that $\mathcal{D}''_{\varphi,P}$ is of size $2^{m^{O(n)} \log m^{O(n)}} = 2^{m^{O(n)}}$ and index $m^{O(n)}$. Finally, solving the parity game can be done in time $(2^{m^{O(n)}})^{m^{O(n)}} = 2^{m^{O(n)}}$. We conclude that the LLTL-noisy-synthesis problem is in 2EXPTIME. Hardness in 2EXPTIME again follows from the hardness of the synthesis problem in the Boolean setting. □ □

Remark 5.4 The approach described in the proofs of Theorems 2.2, 5.1, and 5.3 is *Safraful*, in the sense it involves a construction of a DPW. As has been the case with Boolean synthesis [20], it is possible to proceed Safralessly also in LLTL synthesis with noise. To see this, note that the starting point in Theorem 2.2 can also be a UGCW, and that Lemma 5.2 works with UGCWs. In more details, once we construct a UGCW $\mathcal{U}$ for the specification, possibly with noise incorporated, the Safraless approach expands $\mathcal{U}$ to a universal co-Büchi tree automaton that accepts winning strategies for the system in the corresponding synthesis game, and checks its emptiness. In terms of complexity, rather than paying an additional exponent in the translation of the specification to a deterministic automaton, we pay it in the non-emptiness check of the tree automaton.

5.3 Local stability revisited

In Section 3.2 we studied stability and we have seen that not all LLTL formulas are locally stable (see Theorem 3.6). This gives rise to the question of deciding whether a given LLTL formula is locally stable. In the context of synthesis, if φ is known to be locally stable and we have a transducer $\mathcal{T}$ that realizes $\langle \varphi, P \rangle$ with no noise, we know that $\mathcal{T}$ realizes $\langle \varphi, P \oplus \gamma \rangle$ with noise ν_γ, where $\nu_\gamma(\sigma) = \{\tau : d(\sigma, \tau) \leqslant \gamma\}$, and $P \oplus \gamma$ is the extension of P to noise ν_γ. Thus, $\ell \in P \oplus \gamma$ iff there is $\ell' \in P$ such that $d(\ell, \ell') \leqslant \gamma$.

Theorem 5.5 *Given an* LLTL *formula φ over a lattice $\mathcal{L}$, deciding whether φ is locally stable is PSPACE-complete.*

Proof: In order to show that the problem is in PSPACE, we consider the following, more general, problem: given an LLTL formula φ and a noise-threshold γ, we want to compute the maximal *distraction*, denoted $\Delta_{\varphi,\gamma}$, that noise γ may cause to φ. Formally,

$$\Delta_{\varphi,\gamma} = \max\{d([\![\pi, \varphi]\!], [\![\pi', \varphi]\!]) : \pi, \pi' \in (\mathcal{L}^{AP})^\omega \text{ and } ld(\pi, \pi') \leqslant \gamma\}.$$

Observe that finding $\Delta_{\varphi,\gamma}$ allows us to decide local stability by iterating over all elements $\gamma \in \{1, \ldots, |\mathcal{L}|\}$ and verifying that $\Delta_{\varphi,\gamma} \leqslant \gamma$. Furthermore, in order to compute $\Delta_{\varphi,\gamma}$, it is enough to decide whether $\Delta_{\varphi,\gamma} \leqslant \mu$ for a threshold $\mu \in \{1, ..., |L|\}$, since we can then iterate over thresholds.

We solve the dual problem, namely deciding whether there exist $\pi, \pi' \in (\mathcal{L}^{AP})^\omega$ such that $ld(\pi, \pi') \leqslant \gamma$ and $d([\![\pi, \varphi]\!], [\![\pi', \varphi]\!]) > \mu$. In order to solve this problem, we proceed as follows. In Theorem 4.1 we showed how to how to construct an NGBW $\mathcal{A}_{\varphi,\ell}$ such that $\mathcal{A}_{\varphi,\ell}$ accepts a computation π iff $[\![\pi, \varphi]\!] = \ell$. In Section 5.2, we showed how to construct a

UGCW $\mathcal{D}'_{\varphi,\ell\oplus\mu}$ such that $\mathcal{D}'_{\varphi,\ell\oplus\mu}$ accepts π iff $[\![\pi',\varphi]\!] \in \ell\oplus\mu$ for every computation π' that is ν_γ-indistinguishable from π. Now, there exist $\pi,\pi' \in (\mathcal{L}^{AP})^\omega$ such that $ld(\pi,\pi') \leqslant \gamma$ and $d([\![\pi,\varphi]\!],[\![\pi',\varphi]\!]) > \mu$ iff there exists $\ell \in \mathcal{L}$ such that $[\![\pi,\varphi]\!] = \ell$ and the latter conditions hold. Observe that these conditions hold iff there exists a computation π that is accepted by $\mathcal{A}_{\varphi,\ell}$ but not by $\mathcal{D}'_{\varphi,\ell\oplus\mu}$. Thus, it suffices to decide whether $L(\mathcal{A}_{\varphi,\ell}) \cap \overline{L(\mathcal{D}'_{\varphi,\ell\oplus\mu})} = \varnothing$ for every $\ell \in \mathcal{L}$.

Finally, we analyze the complexity of this procedure. Let $|\mathcal{L}| = m$ and $|\varphi| = n$. Complementation of $\mathcal{D}'_{\varphi,\ell\oplus\mu}$ can be done by constructing $\mathcal{D}'_{\varphi,\overline{\ell\oplus\mu}}$. Hence, both $\mathcal{A}_{\varphi,\ell}$ and $\overline{\mathcal{D}'_{\varphi,\ell\oplus\mu}}$ have $m^{O(n)}$ states. Checking the emptiness of their intersection can be done on-the-fly in PSPACE.

As discussed above, this suggest a PSPACE algorithm for deciding local stability. We now complete the picture by presenting a matching lower bound.

We prove hardness by describing a polynomial time reduction from the satisfiability problem for LTL to the complement of the local-stability problem.

Consider an LTL formula φ over AP. We assume that φ is not valid, thus there is a computation that does not satisfy it (clearly LTL satisfiability is PSPACE-hard also with this promise). We construct an LLTL formula ψ over the lattice $\mathcal{L} = \langle 2^{\{a,b\}}, \subseteq\rangle$ as follows. Let $AP' = \{p' : p \in AP\}$ be a tagged copy of AP. We define $\psi = \varphi \vee \varphi'$ over $AP \cup AP'$, where φ' is obtained form φ by replacing each atomic proposition by its tagged copy. Clearly this reduction is polynomial. We now show that φ is satisfiable iff ψ is not locally stable.

Consider φ as an LLTL formula over $\mathcal{L}$. We observe that if there exists a computation π such that $[\![\pi,\varphi]\!] = \{a\}$, then there exists a computation τ such that $[\![\tau,\varphi]\!] = \{b\}$. Indeed, the lattice $\mathcal{L}$ is symmetric and φ does not contain elements of the form $\{a\}$ or $\{b\}$ to break the symmetry. Thus, we can obtain τ by swapping the roles of a and b in π. From Theorem 3.2 we know that $\mathcal{L}$ is pointed, so in this case there also exists a computation ρ such that $[\![\rho,\varphi]\!] = \{a,b\}$.

We first prove that if φ is not satisfiable, then ψ is locally stable. Observe that if we view φ as an LLTL formula and there exists a computation π such that $[\![\pi,\varphi]\!] = \{a\}$, then φ is satisfiable as an LTL formula. Indeed, a satisfying computation π' for φ can be obtained from π by defining $p \in \pi'_i$ iff $a \in \pi_i(p)$ for all $p \in AP$ and $i \geqslant 0$. Thus, if φ is not satisfiable, then $[\![\pi,\varphi]\!] = \varnothing$ for every computation $\pi \in (\mathcal{L}^{AP})^\omega$, and similarly $[\![\pi',\psi]\!] = \varnothing$ for every $\pi' \in (\mathcal{L}^{AP\cup AP'})^\omega$. Hence, ψ is locally stable.

For the second direction, assume that φ is satisfiable. Thus, there exists a computation $\pi \in (2^{AP})^\omega$ such that $\pi \models \varphi$. By our assumption, there also exists a computation π' such that $\pi' \not\models \varphi$. It is easy to see that by identifying `True` with $\{a,b\}$ and `False` with $\varnothing$, we get $[\![\pi,\varphi]\!] = \{a,b\}$ and $[\![\pi',\varphi]\!] = \varnothing$. For a computation $w \in (\mathcal{L}^{AP})^\omega$, let $\widehat{w} \in (\mathcal{L}^{AP\cup AP'})^\omega$ be the computation obtained from w by copying the behavior of the atoms in AP to their tagged atoms. Thus, for all $i \geqslant 0$, we have $p, p' \in \widehat{w}_i$ iff $p \in w_i$.

Since the maximal distance between elements in $\mathcal{L}$ is 2, there exists a computation τ such that $ld(\pi', \tau) \leqslant 1$ and $ld(\tau, \pi) \leqslant 1$. That is, we can "get" from π' to π by two local changes of 1. From this follows that $ld(\widehat{\pi'}, \widehat{\tau}) \leqslant 1$ and $ld(\widehat{\tau}, \widehat{\pi}) \leqslant 1$. Consider $[\![\tau, \varphi]\!]$. If $[\![\tau, \varphi]\!] = \{a, b\}$, then ψ is not locally stable, since $ld(\widehat{\pi'}, \widehat{\tau}) \leqslant 1$ but $d([\![\widehat{\tau}, \psi]\!], [\![\widehat{\pi'}, \psi]\!]) = 2$. Similarly, if $[\![\tau, \varphi]\!] = \emptyset$, then ψ is not locally stable. Otherwise, w.l.o.g $[\![\tau, \varphi]\!] = \{a\}$. Then, there exists a computation τ' such that $[\![\tau', \varphi]\!] = \{b\}$. The computation τ' is obtained by swapping a and b in τ. Observe that since π' contains only symmetric elements from $\mathcal{L}$ (i.e. $\emptyset$ and $\{a, b\}$), then $ld(\pi', \tau) = ld(\pi', \tau') \leqslant 1$. Finally, since AP and AP' are disjoint, then by assigning τ over AP and τ' over AP', we obtain a computation ρ such that $ld(\widehat{\pi'}, \rho) \leqslant 1$ but $[\![\rho, \psi]\!] = \{a\} \vee \{b\} = \{a, b\}$, and ψ is not locally stable. □ □

References

[1] S. Almagor, U. Boker, and O. Kupferman. Formalizing and reasoning about quality. In *Proc. 40th Int. Colloq. on Automata, Languages, and Programming*, volume 7966 of *Lecture Notes in Computer Science*, pages 15–27. Springer, 2013.

[2] R. Alur, A. Kanade, and G. Weiss. Ranking automata and games for prioritized requirements. In *Proc. 20th Int. Conf. on Computer Aided Verification*, volume 5123 of *Lecture Notes in Computer Science*, pages 240–253. Springer, 2008.

[3] R. Bloem, K. Chatterjee, T. Henzinger, and B. Jobstmann. Better quality in synthesis through quantitative objectives. In *Proc. 21st Int. Conf. on Computer Aided Verification*, volume 5643 of *Lecture Notes in Computer Science*, pages 140–156. Springer, 2009.

[4] P. Cerný and T. Henzinger. From boolean to quantitative synthesis. In *EMSOFT*, pages 149–154, 2011.

[5] K. Chatterjee, L. Doyen, and T. Henzinger. Quantative languages. In *Proc. 17th Annual Conf. of the European Association for Computer Science Logic*, pages 385–400, 2008.

[6] K. Chatterjee, L. Doyen, T. A. Henzinger, and J.-F. Raskin. Algorithms for omega-regular games with imperfect information. In *Proc. 15th Annual Conf. of the European Association for Computer Science Logic*, volume 4207 of *Lecture Notes in Computer Science*, pages 287–302, 2006.

[7] K. Chatterjee and R. Majumdar. Minimum attention controller synthesis for omega-regular objectives. In *FORMATS*, pages 145–159, 2011.

[8] K. Chatterjee, R. Majumdar, and T. A. Henzinger. Controller synthesis with budget constraints. In *Proc 11th International Workshop on Hybrid Systems: Computation and Control*, volume 4981 of *Lecture Notes in Computer Science*, pages 72–86. Springer, 2008.

[9] M. Chechik, B. Devereux, and A. Gurfinkel. Model-checking infinite state-space systems with fine-grained abstractions using SPIN. In *Proc. 8th Int. SPIN Workshop on Model Checking Software*, volume 2057 of *Lecture Notes in Computer Science*, pages 16–36. Springer, 2001.

[10] A. Church. Logic, arithmetics, and automata. In *Proc. Int. Congress of Mathematicians, 1962*, pages 23–35. Institut Mittag-Leffler, 1963.

[11] E. Emerson and C. Jutla. Tree automata, μ-calculus and determinacy. In *Proc. 32nd IEEE Symp. on Foundations of Computer Science*, pages 368–377, 1991.

[12] M. Faella, A. Legay, and M. Stoelinga. Model checking quantitative linear time logic. *Electr. Notes Theor. Comput. Sci.*, 220(3):61–77, 2008.

[13] E. Filiot, N. Jin, and J.-F. Raskin. An antichain algorithm for LTL realizability. In *Proc. 21st Int. Conf. on Computer Aided Verification*, volume 5643, pages 263–277, 2009.

[14] M. Huth and S. Pradhan. Consistent partial model checking. *Electr. Notes Theor. Comput. Sci.*, 73:45–85, 2004.

[15] M. Jurdzinski, M. Paterson, and U. Zwick. A deterministic subexponential algorithm for solving parity games. *SIAM Journal on Computing*, 38(4):1519–1532, 2008.

[16] R. Kumar and M. Shayman. Supervisory control of nondeterministic systems under partial observation and decentralization. *SIAM Journal of Control and Optimization*, 1995.

[17] O. Kupferman and Y. Lustig. Lattice automata. In *Proc. 8th Int. Conf. on Verification, Model Checking, and Abstract Interpretation*, volume 4349 of *Lecture Notes in Computer Science*, pages 199 – 213. Springer, 2007.

[18] O. Kupferman and Y. Lustig. Latticed simulation relations and games. *International Journal on the Foundations of Computer Science*, 21(2):167–189, 2010.

[19] O. Kupferman and M. Vardi. Church's problem revisited. *The Bulletin of Symbolic Logic*, 5(2):245 – 263, 1999.

[20] O. Kupferman and M. Vardi. Safraless decision procedures. In *Proc. 46th IEEE Symp. on Foundations of Computer Science*, pages 531–540, 2005.

[21] M. Kwiatkowska. Quantitative verification: models techniques and tools. In *ESEC/SIGSOFT FSE*, pages 449–458, 2007.

[22] D. Martin. Borel determinacy. *Annals of Mathematics*, 65:363–371, 1975.

[23] S. Miyano and T. Hayashi. Alternating finite automata on ω-words. *Theoretical Computer Science*, 32:321–330, 1984.

[24] N. Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. In *Proc. 21st IEEE Symp. on Logic in Computer Science*, pages 255–264. IEEE press, 2006.

[25] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. 16th ACM Symp. on Principles of Programming Languages*, pages 179–190, 1989.

[26] A. Pnueli and R. Rosner. On the synthesis of an asynchronous reactive module. In *Proc. 16th Int. Colloq. on Automata, Languages, and Programming*, volume 372 of *Lecture Notes in Computer Science*, pages 652–671. Springer, 1989.

[27] J. Reif. The complexity of two-player games of incomplete information. *Journal of Computer and Systems Science*, 29:274–301, 1984.

[28] S. Safra. Exponential determinization for ω-automata with strong-fairness acceptance condition. In *Proc. 24th ACM Symp. on Theory of Computing*, 1992.

[29] S. Schewe. Solving parity games in big steps. In *Proc. 27th Conf. on Foundations of Software Technology and Theoretical Computer Science*, pages 449–460, 2007.

[30] M. Vardi. From verification to synthesis. In *Proc. 2nd International Conference on Verified Software: Theories, Tools, Experiments*, volume 5295 of *Lecture Notes in Computer Science*, page 2. Springer, 2008.

[31] M. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.

[32] Y. Velner and A. Rabinovich. Church synthesis problem for noisy input. In *Proc. 14th Int. Conf. on Foundations of Software Science and Computation Structures*, pages 275–289, 2011.

Regular Sensing[*]

Shaull Almagor[†] *Denis Kuperberg* [‡] *Orna Kupferman* [§]

Abstract

The size of deterministic automata required for recognizing regular and ω-regular languages is a well-studied measure for the complexity of languages. We introduce and study a new complexity measure, based on the *sensing* required for recognizing the language. Intuitively, the sensing cost quantifies the detail in which a random input word has to be read in order to decide its membership in the language. We show that for finite words, size and sensing are related, and minimal sensing is attained by minimal automata. Thus, a unique minimal-sensing deterministic automaton exists, and is based on the language's right-congruence relation. For infinite words, the minimal sensing may be attained only by an infinite sequence of automata. We show that the optimal limit cost of such sequences can be characterized by the language's right-congruence relation, which enables us to find the sensing cost of ω-regular languages in polynomial time.

1 Introduction

Studying the complexity of a formal language, there are several complexity measures to consider. When the language is given by means of a Turing Machine, the traditional measures are time and space demands. Theoretical interest as well as practical considerations have motivated additional measures, such as randomness (the number of random bits required for the execution) [10] or communication complexity (number and length of messages required) [9]. For regular and ω-regular languages, given by means of finite-state automata, the classical complexity measure is the size of a minimal deterministic automaton that recognizes the language.

We introduce and study a new complexity measure, namely the *sensing cost* of the language. Intuitively, the sensing cost of a language measures the detail with which a random input word needs to be read in order to decide membership in the language. Sensing has been studied in several other CS contexts. In theoretical CS, in methodologies

[*] Published in the proceedings of the 34th International Conference on Foundation of Software Technology and Theoretical Computer Science, LIPIcs 29, pages 161–173, Schloss Dagstuhl, 2014.

[†] The Hebrew University, Jerusalem, Israel.

[‡] The University of Warsaw, Faculty of Mathematics, Informatics, and Mechanics.

[§] The Hebrew University, Jerusalem, Israel.

such as PCP and property testing, we are allowed to sample or query only part of the input [7]. In more practical applications, mathematical tools in signal processing are used to reconstruct information based on compressed sensing [5], and in the context of data streaming, one cannot store in memory the entire input, and therefore has to approximate its properties according to partial "sketches" [11].

Our interest in regular sensing is motivated by the use of finite-state automata (as well as monitors, controllers, and transducers) in reasoning about on-going behaviors of reactive systems. In particular, a big challenge in the design of monitors is an optimization of the sensing needed for deciding the correctness of observed behaviors. Our goal is to formalize regular sensing in the finite-state setting and to study the sensing complexity measure for regular and ω-regular languages.

A natural setting in which sensing arises is *synthesis*: given a specification over sets I and O of input and output signals, the goal is to construct a finite-state system that, given a sequence of input signals, generates a computation that satisfies the specification. In each moment in time, the system reads an assignment to the input signals, namely a letter in 2^I, which requires the activation of $|I|$ Boolean sensors. A well-studied special case of limited sensing is synthesis with *incomplete information*. There, the system can read only a subset of the signals in I, and should still generate only computations that satisfy the specification [8, 3]. A more sophisticated case of sensing in the context of synthesis is studied in [4], where the system can read some of the input signals some of the time. In more detail, sensing the truth value of an input signal has a cost, the system has a budget for sensing, and it tries to realize the specification while minimizing the required sensing budget.

We study the fundamental questions on regular sensing. We consider languages over alphabets of the form 2^P, for a finite set P of signals. Consider a deterministic automaton $\mathcal{A}$ over an alphabet 2^P. For a state q of $\mathcal{A}$, we say that a signal $p \in P$ is *sensed* in q if at least one transition taken from q depends on the truth value of p. The *sensing cost* of q is the number of signals it senses, and the sensing cost of a run is the average sensing cost of states visited along the run. We extend the definition to automata by assuming a uniform distribution of the inputs.[1] Thus, the sensing cost of $\mathcal{A}$ is the limit of the expected sensing of runs over words of increasing length.[2] We show that this definition coincides

[1] Our study and results apply also to a non-uniform distribution on the letters, given by a Markov chain (see Remark 4.8).

[2] Alternatively, one could define the sensing cost of $\mathcal{A}$ as the cost of its "most sensing" run. Such a worst-case approach is taken in [4], where the sensing cost needs to be kept under a certain budget in all computations, rather than in expectation. We find the average-case approach we follow appropriate for sensing, as the cost of operating sensors may well be amortized over different runs of the system, and requiring the budget to be kept under a threshold in every run may be too restrictive. Thus, the automaton must answer correctly for every word, but the sensing should be low only on average, and it is allowed to operate an expensive sensor now and then.

with one that is based on the stationary distribution of the Markov chain induced by $\mathcal{A}$, which enables us to calculate the sensing cost of an automaton in polynomial time. The sensing cost of a language L, of either finite or infinite words, is then the infimum of the sensing costs of deterministic automata for L. In the case of infinite words, one can study different classes of automata, yet we show that the sensing cost is independent of the acceptance condition being used.

We start by studying the sensing cost of regular languages of finite words. For the complexity measure of size, the picture in the setting of finite words is very clean: each language L has a unique minimal deterministic automaton (DFA), namely the *residual automaton* $\mathcal{R}_L$ whose states correspond to the equivalence classes of the Myhill-Nerode right-congruence relation for L. We show that minimizing the state space of a DFA can only reduce its sensing cost. Hence, the clean picture of the size measure is carried over to the sensing measure: the sensing cost of a language L is attained in the DFA $\mathcal{R}_L$. In particular, since DFAs can be minimized in polynomial time, we can construct in polynomial time a minimally-sensing DFA, and can compute in polynomial time the sensing cost of languages given by DFAs.

We then study the sensing cost of ω-regular languages, given by means of deterministic parity automata (DPAs). Recall the size complexity measure. There, the picture for languages of infinite words is not clean: A language needs not have a unique minimal DPA, and the problem of finding one is NP-complete [13]. It turns out that the situation is challenging also in the sensing measure. First, we show that different minimal DPAs for a language may have different sensing costs. In fact, bigger DPAs may have smaller sensing costs.

Before describing our results, let us describe a motivating example that demonstrates the intricacy in the case of ω-regular languages. Consider a component in a vacuum-cleaning robot that monitors the dust collector and checks that it is empty infinitely often. The proposition $empty$ indicates whether the collector is empty and a sensor needs to be activated in order to know its truth value. One implementation of the component would sense $empty$ throughout the computation. This corresponds to the classical two-state DPA for "infinitely often $empty$". A different implementation can give up the sensing of $empty$ for some fixed number k of states, then wait for $empty$ to hold, and so forth. The bigger k is, the lazier is the sensing and the smaller the sensing cost is. As the example demonstrates, there may be a trade-off between the sensing cost of an implementation and its size. Other considerations, like a preference to have eventualities satisfied as soon as possible, enter the picture too.

Our main result is that despite the above intricacy, the sensing cost of an ω-regular language L is the sensing cost of the residual automaton $\mathcal{R}_L$ for L. It follows that the sensing cost of an ω-regular language can be computed in polynomial time. Unlike the case of finite words, it may not be possible to define L on top of $\mathcal{R}_L$. Interestingly, however, $\mathcal{R}_L$ does capture exactly the sensing required for recognizing L. The proof of this property of $\mathcal{R}_L$ is the main technical challenge of our contribution. The proof goes via a sequence $(\mathcal{B}_n)_{n=1}^{\infty}$ of DPAs whose sensing costs converge to that of L. The DPA $\mathcal{B}_n$ is obtained from a DPA $\mathcal{A}$ for L by a lazy sensing strategy that spends time in n copies of $\mathcal{R}_L$ between visits to $\mathcal{A}$, but spends enough time in $\mathcal{A}$ to ensure that the language is L. It is worth noting that this result is far from being intuitive. Indeed, first, as mentioned above, the extra expressive power that is added to the setting by the acceptance condition of DPAs makes the residual automaton irrelevant in the context of size minimization. Moreover, in the context of sensing, there need not be a single DPA that attains the minimal sensing cost. It is thus surprising that $\mathcal{R}_L$, which has no acceptance condition, captures the sensing cost of all DPAs. We believe that this reflects a general property of deterministic parity automata that could be useful outside of the scope of sensing. Intuitively, it means that we can "lose track" of the run of a deterministic automaton for arbitrary long periods, just keeping the residual in memory, and still be able to recognize the wanted language.

Due to lack of space, some of the proofs can be found in the Appendix.

2 Preliminaries

Automata.

A *deterministic automaton on finite words* (DFA, for short) is $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, \alpha \rangle$, where Q is a finite set of states, $q_0 \in Q$ is an initial state, $\delta : Q \times \Sigma \to Q$ is a total transition function, and $\alpha \subseteq Q$ is a set of accepting states. We sometimes refer to δ as a relation $\Delta \subseteq Q \times \Sigma \times Q$, with $\langle q, \sigma, q' \rangle \in \Delta$ iff $\delta(q, \sigma) = q'$. The run of $\mathcal{A}$ on a word $w = \sigma_1 \cdot \sigma_2 \cdots \sigma_m \in \Sigma^*$ is the sequence of states $q_0, q_1, \ldots, q_m$ such that $q_{i+1} = \delta(q_i, \sigma_{i+1})$ for all $i \geqslant 0$. The run is accepting if $q_m \in \alpha$. A word $w \in \Sigma^*$ is accepted by $\mathcal{A}$ if the run of $\mathcal{A}$ on w is accepting. The language of $\mathcal{A}$, denoted $L(\mathcal{A})$, is the set of words that $\mathcal{A}$ accepts. For a state $q \in Q$, we use $\mathcal{A}^q$ to denote $\mathcal{A}$ with initial state q. We sometimes refer also to nondeterministic automata (NFAs), where $\delta : Q \times \Sigma \to 2^Q$ suggests several possible successor states. Thus, an NFA may have several runs on an input word w, and it accepts w if at least one of them is accepting.

Consider a language $L \subseteq \Sigma^*$. For two finite words u_1 and u_2, we say that u_1 and u_2 are *right L-indistinguishable*, denoted $u_1 \sim_L u_2$, if for every $z \in \Sigma^*$, we have that $u_1 \cdot z \in L$ iff $u_2 \cdot z \in L$. Thus, $\sim_L$ is the Myhill-Nerode right congruence used for minimizing automata. For $u \in \Sigma^*$, let $[u]$ denote the equivalence class of u in $\sim_L$ and let $\langle L \rangle$ denote the set of all equivalence classes. Each class $[u] \in \langle L \rangle$ is associated with the *residual language* $u^{-1}L = \{w : uw \in L\}$. When L is regular, the set $\langle L \rangle$ is finite, and induces the *residual automaton* of L, defined by $\mathcal{R}_L = \langle \Sigma, \langle L \rangle, \Delta_L, [\epsilon], \alpha \rangle$, with $\langle [u], a, [u \cdot a] \rangle \in \Delta_L$ for all $[u] \in \langle L \rangle$ and $a \in \Sigma$. Also, α contains all classes $[u]$ with $u \in L$. The DFA $\mathcal{R}_L$ is well defined and is the unique minimal DFA for L.

A *deterministic automaton on infinite words* is $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, \alpha \rangle$, where Q, q_0, and δ are as in DFA, and α is an acceptance condition. The run of $\mathcal{A}$ on an infinite input word $w = \sigma_1 \cdot \sigma_2 \cdots \in \Sigma^\omega$ is defined as for automata on finite words, except that the sequence of visited states is now infinite. For a run $r = q_0, q_1, \ldots$, let $\mathit{inf}(r)$ denote the set of states that r visits infinitely often. Formally, $\mathit{inf}(r) = \{q : q = q_i$ for infinitely many i's$\}$. We consider the following acceptance conditions. In a *Büchi* automaton, the acceptance condition is a set $\alpha \subseteq Q$ and a run r is accepting iff $\mathit{inf}(r) \cap \alpha \neq \emptyset$. Dually, in a *co-Büchi*, again $\alpha \subseteq Q$, but r is accepting iff $\mathit{inf}(r) \cap \alpha = \emptyset$. Finally, parity condition is a mapping $\alpha : Q \to [i, \ldots, j]$, for integers $i \leqslant j$, and a run r is accepting iff $\max_{q \in \mathit{inf}(r)}\{\alpha(q)\}$ is even.

We extend the right congruence $\sim_L$ as well as the definition of the residual automaton $\mathcal{R}_L$ to languages $L \subseteq \Sigma^\omega$. Here, however, $\mathcal{R}_L$ need not accept the language of L, and we ignore its acceptance condition.

Sensing.

We study languages over an alphabet $\Sigma = 2^P$, for a finite set P of signals. A letter $\sigma \in \Sigma$ corresponds to a truth assignment to the signals. When we define languages over Σ, we use predicates on P in order to denote sets of letters. For example, if $P = \{a, b, c\}$, then the expression $(\texttt{True})^* \cdot a \cdot b \cdot (\texttt{True})^*$ describes all words over 2^P that contain a subword $\sigma_a \cdot \sigma_b$ with $\sigma_a \in \{\{a\}, \{a, b\}, \{a, c\}, \{a, b, c\}\}$ and $\sigma_b \in \{\{b\}, \{a, b\}, \{b, c\}, \{a, b, c\}\}$.

Consider an automaton $\mathcal{A} = \langle 2^P, Q, q_0, \delta, \alpha \rangle$. For a state $q \in Q$ and a signal $p \in P$, we say that p is *sensed in* q if there exists a set $S \subseteq P$ such that $\delta(q, S \backslash \{p\}) \neq \delta(q, S \cup \{p\})$. Intuitively, a signal is sensed in q if knowing its value may affect the destination of at least one transition from q. We use $\mathit{sensed}(q)$ to denote the set of signals sensed in q. The *sensing cost* of a state $q \in Q$ is $\mathit{scost}(q) = |\mathit{sensed}(q)|$. [3]

Consider a deterministic automaton $\mathcal{A}$ over $\Sigma = 2^P$ (and over finite or infinite words). For a finite run $r = q_1, \ldots, q_m$ of $\mathcal{A}$, we define the sensing cost of r, denoted $\mathit{scost}(r)$, as $\frac{1}{m}\sum_{i=1}^{m} \mathit{scost}(q_i)$. That is, $\mathit{scost}(r)$ is the average number of sensors that $\mathcal{A}$ uses during r. Now, for a finite word w, we define the sensing cost of w in $\mathcal{A}$, denoted $\mathit{scost}_{\mathcal{A}}(w)$, as the sensing cost of the run of $\mathcal{A}$ on w. Finally, the sensing cost of $\mathcal{A}$ is the expected sensing cost of words of length that tends to infinity, where we assume that the letters in Σ are uniformly distributed. Thus, $\mathit{scost}(\mathcal{A}) = \lim_{m \to \infty} |\Sigma|^{-m} \sum_{w:|w|=m} \mathit{scost}_{\mathcal{A}}(w)$. Note that the definition applies to automata on both finite and infinite words.

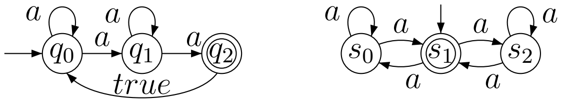
Two DFAs may recognize the same language and have different sensing costs. In

[3] We note that, alternatively, one could define the *sensing level* of states, with $\mathit{slevel}(q) = \frac{|\mathit{sensed}(q)|}{|P|}$. Then, for all states q, we have that $\mathit{slevel}(q) \in [0, 1]$. All our results hold also for this definition, simply by dividing the sensing cost by $|P|$.

fact, as we demonstrate in Example 2.1 below, in the case of infinite words two different minimal automata for the same language may have different sensing costs.

For a language L of finite or infinite words, the sensing cost of L, denoted $scost(L)$ is the minimal sensing cost required for recognizing L by a deterministic automaton. Thus, $scost(L) = \inf_{\mathcal{A}:L(\mathcal{A})=L} scost(\mathcal{A})$. For the case of infinite words, we allow $\mathcal{A}$ to be a deterministic automaton of any type. In fact, as we shall see, unlike the case of succinctness, the sensing cost is independent of the acceptance condition used.

Example 2.1 *Let $P = \{a\}$. Consider the language $L \subseteq (2^{\{a\}})^\omega$ of all words with infinitely many a and infinitely many $\neg a$. In the following figure we present two minimal DBAs (deterministic Büchi automata) for L with different sensing costs.*



While all the states of the second automaton sense a, thus its sensing cost is 1, the signal a is not sensed in all the states of the first automaton, thus its sensing cost is strictly smaller than 1 (to be precise, it is $\frac{4}{5}$, as we shall see in Example 2.2).

Remark 2.1 Our study of sensing considers deterministic automata. The notion of sensing is less natural in the nondeterministic setting. From a conceptual point of view, we want to capture the number of sensors required for an actual implementation for recognizing the language. Technically, guesses can reduce the number of required sensors. To see this, take $P = \{a\}$ and consider the language $L = \texttt{True}^* \cdot a$. A DFA for L needs two states, both sensing a. An NFA for L can guess the position of the letter before the last one, where it moves to the only state that senses a. The sensing cost of such an NFA is 0 (for any reasonable extension of the definition of cost on NFAs). □

Probability.

Consider a directed graph $G = \langle V, E \rangle$. A *strongly connected component* (SCC) of G is a maximal (with respect to containment) set $C \subseteq V$ such that for all $x, y \in C$, there is a path from x to y. An SCC (or state) is *ergodic* if no other SCC is reachable from it, and is *transient* otherwise.

An automaton $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, \alpha \rangle$ induces a directed graph $G_{\mathcal{A}} = \langle Q, E \rangle$ in which $\langle q, q' \rangle \in E$ iff there is a letter σ such that $q' \in \delta(q, \sigma)$. When we talk about the SCCs of $\mathcal{A}$, we refer to those of $G_{\mathcal{A}}$. Recall that we assume that the letters in Σ are uniformly distributed, thus $\mathcal{A}$ also corresponds to a Markov chain $M_{\mathcal{A}}$ in which the probability of a transition from state q to state q' is $p_{q,q'} = \frac{1}{|\Sigma|}|\{\sigma \in \Sigma : \delta(q, \sigma) = q'\}|$. Let $\mathcal{C}$ be the set of $\mathcal{A}$'s SCC, and $\mathcal{C}_e \subseteq \mathcal{C}$ be the set of its ergodic SCC's.

Consider an ergodic SCC $C \in \mathcal{C}_e$. Let P_C be the matrix describing the probability of transitions in C. Thus, the rows and columns of P_C are associated with states, and the value in coordinate q, q' is $p_{q,q'}$. By [6], there is a unique probability vector $\pi_C \in [0,1]^C$ such that $\pi_C P_C = \pi_C$. This vector describes the *stationary distribution* of C: for all $q \in C$ it holds that $\pi_C(q) = \lim_{m\to\infty} \frac{E^C_m(q)}{m}$, where $E^C_m(q)$ is the average number of occurrences of q in a run of M_A of length m that starts anywhere in C [6]. Thus, intuitively, $\pi_C(q)$ is the probability that a long run that starts in C ends in q. In order to extend the distribution to the entire Markov chain of $\mathcal{A}$, we have to take into account the probability of reaching each of the ergodic components. The *SCC-reachability distribution* of $\mathcal{A}$ is the function $\rho : \mathcal{C} \to [0,1]$ that maps each ergodic SCC C of $\mathcal{A}$ to the probability that M_A eventually reaches C, starting from the initial state. We can now define the *limiting distribution* $\pi : Q \to [0,1]$, as

$$\pi(q) = \begin{cases} 0 & \text{if } q \text{ is transient,} \\ \pi_C(q)\rho(C) & \text{if } q \text{ is in some } C \in \mathcal{C}_e. \end{cases}$$

Note that $\sum_{q\in Q} \pi(q) = 1$, and that if P is the matrix describing the transitions of M_A and π is viewed as a vector in $[0,1]^Q$, then $\pi P = \pi$. Intuitively, the limiting distribution of state q describes the probability of a run on a random and long input word to end in q. Formally, we have the following lemma, whose proof appears in Appendix A.1.

Lemma 2.2 *Let $E_m(q)$ be the expected number of occurrences of a state q in a run of length m of M_A that starts in q_0. Then, $\pi(q) = \lim_{m\to\infty} \frac{E_m(q)}{m}$.*

Computing The Sensing Cost of an Automaton.

Consider a deterministic automaton $\mathcal{A} = \langle 2^P, Q, \delta, q_0, \alpha\rangle$. The definition of $scost(\mathcal{A})$ by means of the expected sensing cost of words of length that tends to infinity does not suggest an algorithm for computing it. In this section we show that the definition coincides with a definition that sums the costs of the states in $\mathcal{A}$, weighted according to the limiting distribution, and show that this implies a polynomial-time algorithm for computing $scost(\mathcal{A})$. This also shows that the cost is well-defined for all automata.

Theorem 2.3 *For all automata $\mathcal{A}$, we have $scost(\mathcal{A}) = \sum_{q\in Q} \pi(q) \cdot scost(q)$, where π is the limiting distribution of $\mathcal{A}$.*

Remark 2.4 *It is not hard to see that if $\mathcal{A}$ is strongly connected, then π is the unique stationary distribution of M_A and is independent of the initial state of $\mathcal{A}$. Accordingly, $scost(\mathcal{A})$ is also independent of $\mathcal{A}$'s initial state in this special case.* □

Theorem 2.5 *Given an automaton $\mathcal{A}$, the sensing cost $scost(\mathcal{A})$ can be calculated in polynomial time.*

Proof: By Theorem 2.3, we have that $scost(\mathcal{A}) = \sum_{q \in Q} \pi(q) \cdot scost(q)$, where π is the limiting distribution of $\mathcal{A}$. By the definition of π, we have that $\pi(q) = \pi_C(q)\rho(C)$, if q is in some $C \in \mathcal{C}_e$. Otherwise, $\pi(q) = 0$. Hence, the computational bottleneck is the calculation of the SCC-reachability distribution $\rho : C \rightarrow [0, 1]$ and the stationary distributions π_C for every $C \in \mathcal{C}_e$. It is well known that both can be computed in polynomial time via classic algorithms on matrices. For completeness, we give the details in Appendix A.3. □

Example 2.2 *Recall the first DBA described in Example 2.1. Its limiting distribution is* $\pi(q_0) = \pi(q_1) = \frac{2}{5}$, $\pi(q_2) = \frac{1}{5}$. *Accordingly, its cost is* $1 \cdot \frac{2}{5} + 1 \cdot \frac{2}{5} + 0 \cdot \frac{1}{5} = \frac{4}{5}$.

Additional examples can be found in Appendix A.4.

3 The Sensing Cost of Regular Languages of Finite Words

In this section we study the setting of finite words. We show that there, sensing minimization goes with size minimization, which makes things clean and simple, as size minimization for DFAs is a feasible and well-studied problem. We also study theoretical properties of sensing. We show that, surprisingly, abstraction of signals may actually increase the sensing cost of a language, and we study the effect of classical operations on regular languages on their sensing cost. These last two contributions can be found in Appendices A.9 and A.10.

Consider a regular language $L \subseteq \Sigma^*$, with $\Sigma = 2^P$. Recall that the residual automaton $\mathcal{R}_L = \langle \Sigma, \langle L \rangle, \Delta_L, [\epsilon], \alpha \rangle$ is the minimal-size DFA that recognizes L. We claim that $\mathcal{R}_L$ also minimizes the sensing cost of L.

Lemma 3.1 *Consider a regular language* $L \subseteq \Sigma^*$. *For every DFA* $\mathcal{A}$ *with* $L(\mathcal{A}) = L$, *we have that* $scost(\mathcal{A}) \geqslant scost(\mathcal{R}_L)$.

Proof: Consider a word $u \in \Sigma^*$. After reading u, the DFA $\mathcal{R}_L$ reaches the state $[u]$ and the DFA $\mathcal{A}$ reaches a state q with $L(\mathcal{A}^q) = u^{-1}L$. Indeed, otherwise we can point to a word with prefix u that is accepted only in one of the DFAs. We claim that for every state $q \in Q$ such that $L(\mathcal{A}^q) = u^{-1}L$, it holds that $sensed([u]) \subseteq sensed(q)$. To see this, consider a signal $p \in sensed([u])$. By definition, there exists a set $S \subseteq P$ and words u_1 and u_2 such that $([u], S \backslash \{p\}, [u_1]) \in \Delta_L$, $([u], S \cup \{p\}, [u_2]) \in \Delta_L$, yet $[u_1] \neq [u_2]$. By the definition of $\mathcal{R}_L$, there exists $z \in (2^P)^*$ such that, w.l.o.g, $z \in u_1^{-1}L \backslash u_2^{-1}L$. Hence, as $L(\mathcal{A}^q) = u^{-1}L$, we have that $\mathcal{A}^q$ accepts $(S \backslash \{p\}) \cdot z$ and rejects $(S \cup \{p\}) \cdot z$. Let $\delta_{\mathcal{A}}$ be the transition function of $\mathcal{A}$. By the above, $\delta_{\mathcal{A}}(q, S \backslash \{p\}) \neq \delta_{\mathcal{A}}(q, S \cup \{p\})$. Therefore, $p \in sensed(q)$, and we are done. Now, $sensed([u]) \subseteq sensed(q)$ implies that $scost(q) \geqslant scost([u])$.

Consider a word $w_1 \cdots w_m \in \Sigma^*$. Let $r = r_0, \ldots, r_m$ and $[u_0], \ldots, [u_m]$ be the runs of $\mathcal{A}$ and $\mathcal{R}_L$ on w, respectively. Note that for all $i \geqslant 0$, we have $u_i = w_1 \cdot w_2 \cdots w_i$.

For all $i \geqslant 0$, we have that $L(\mathcal{A}^{r_i}) = u_i^{-1}L$, implying that then $scost(r_i) \geqslant scost([u_i])$. Hence, $scost_{\mathcal{A}}(w) \geqslant scost_{\mathcal{R}_L}(w)$. Since this holds for every word in Σ^*, it follows that $scost(\mathcal{A}) \geqslant scost(\mathcal{R}_L)$. □

Since $L(\mathcal{R}_L) = L$, then $scost(L) \leqslant scost(\mathcal{R}_L)$. This, together with Lemma 3.1, enables us to conclude the following.

Theorem 3.2 *For every regular language $L \subseteq \Sigma^*$, we have $scost(L) = scost(\mathcal{R}_L)$.*

Finally, since DFAs can be size-minimized in polynomial time, Theorems 2.5 and 3.2 imply we can efficiently minimize also the sensing cost of a DFA and calculate the sensing cost of its language:

Theorem 3.3 *Given a DFA $\mathcal{A}$, the problem of computing $scost(L(\mathcal{A}))$ can be solved in polynomial time.*

4 The Sensing Cost of ω-Regular Languages

For the case of finite words, we have a very clean picture: minimizing the state space of a DFA also minimizes its sensing cost. In this section we study the case of infinite words. There, the picture is much more complicated. In Example 2.1 we saw that different minimal DBAs may have a different sensing cost. We start this section by showing that even for languages that have a single minimal DBA, the sensing cost may not be attained by this minimal DBA, and in fact it may be attained only as a limit of a sequence of DBAs.

Example 4.1 *Let $P = \{p\}$, and consider the language L of all words $w_1 \cdot w_2 \cdots$ such that $w_i = \{p\}$ for infinitely many i's. Thus, $L = (\texttt{True}^* \cdot p)^\omega$. A minimal DBA for L has two states. The minimal sensing cost for a two-state DBA for L is $\frac{2}{3}$ (the classical two-state DBA for L senses p in both states and thus has sensing cost 1. By taking $\mathcal{A}_1$ in the sequence we shall soon define we can recognize L by a two-state DBA with sensing cost $\frac{2}{3}$). Consider the sequence of DBAs $\mathcal{A}_m$ appearing in Figure 1. The DBA $\mathcal{A}_m$ recognizes $(\texttt{True}^{\geqslant m} \cdot p)^\omega$, which is equivalent to L, yet enables a "lazy" sensing of p. Formally, the stationary distribution π for $\mathcal{A}_m$ is such that $\pi(q_i) = \frac{1}{m+2}$ for $0 \leqslant i \leqslant m-1$ and $\pi(q_m) = \frac{2}{m+2}$. In the states $q_0, \ldots, q_{m-1}$ the sensing cost is 0 and in q_m it is 1. Accordingly, $scost(\mathcal{A}_m) = \frac{2}{m+2}$, which tends to 0 as m tends to infinity.*

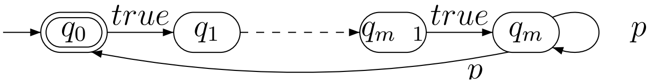


Figure 1: The DBA $\mathcal{A}_m$.

4.1 Characterizing $scost(L)$ by the residual automaton for L

In this section we state and prove our main result, which characterizes the sensing cost of an ω-regular language by means of the residual automaton for the language:

Theorem 4.1 *For every ω-regular language $L \subseteq \Sigma^\omega$, we have $scost(L) = scost(\mathcal{R}_L)$.*

The proof is described over the following section. The first direction, showing that $scost(L) \geqslant scost(\mathcal{R}_L)$, is proved by similar considerations to those used in the proof of Lemma 3.1 for the setting of finite words, and can be found in Appendix A.5.

Our main effort is to prove that $scost(L) \leqslant scost(\mathcal{R}_L)$. To show this, we construct, given a DPA $\mathcal{A}$ such that $L(\mathcal{A}) = L$, a sequence $(\mathcal{B}_n)_{n\geqslant 1}$ of DPAs such that $L(\mathcal{B}_n) = L$ for every $n \geqslant 1$, and $\lim_{n\to\infty} scost(\mathcal{B}_n) = scost(\mathcal{R}_L)$. We note that since the DPAs $\mathcal{B}_n$ have the same acceptance condition as $\mathcal{A}$, there is no trade-off between sensing cost and acceptance condition. More precisely, if L can be recognized by a DPA with parity ranks $[i, j]$ (in particular, if L is DBA-recognizable), then the sensing cost for $L(\mathcal{A})$ can be obtained by a DPA with parity ranks $[i, j]$.

We first assume that $\mathcal{A}$ is strongly connected. We will later show how to drop this assumption.

Let $\mathcal{A} = \langle \Sigma, Q, q_0, \Delta, \alpha_{\mathcal{A}} \rangle$ be a strongly connected DPA for L. We assume that $\mathcal{A}$ is minimally ranked. Thus, if $\mathcal{A}$ has parity ranks $\{0, 1, \ldots, k\}$, then there is no DPA for L with ranks $\{0, 1, \ldots, k-1\}$ or $\{1, 2, \ldots, k\}$. Also, if $\mathcal{A}$ has ranks $\{1, 2, \ldots, k\}$, we consider the complement DPA, which is $\mathcal{A}$ with ranks $\{0, 1, \ldots, k-1\}$. Since DPAs can be complemented by dualizing the acceptance condition, their sensing cost is preserved under complementation, so reasoning about the complemented DPA is sound. For $0 \leqslant i \leqslant k$, a cycle in $\mathcal{A}$ is called an *i-loop* if the maximal rank along the cycle is i. For $0 \leqslant i \leqslant j \leqslant k$, an *$[i, j]$-flower* is a state $q_{❀} \in Q$ such that for every $i \leqslant r \leqslant j$, there is an r-loop that goes through $q_{❀}$.

The following is an adaptation of a result from [12] to strongly connected DPAs:

Lemma 4.2 *Consider a strongly-connected minimally-ranked DPA $\mathcal{A} = \langle \Sigma, Q, q_0, \Delta, \alpha_{\mathcal{A}} \rangle$ with ranks $\{0, \ldots, k\}$. Then, there is a DPA $\mathcal{D} = \langle \Sigma, Q, q_0, \Delta, \alpha_{\mathcal{D}} \rangle$ such that all the following hold.*

1. *For every state $s \in Q$, we have $L(\mathcal{A}^s) = L(\mathcal{D}^s)$. In particular, $\mathcal{A}$ and $\mathcal{D}$ are equivalent.*

2. *There exists $m \in \mathbb{N}$ such that $\mathcal{D}$ has ranks $\{0, ..., 2m + k\}$ and has a $[2m, 2m + k]$ flower.*

Proof: We start with the following claim, whose proof appears in Appendix A.6.

Claim 4.3 *$\mathcal{A}$ does not have an equivalent DPA with ranks $\{1, \ldots, k + 1\}$.*

Now, [12] proves the lemma for $\mathcal{A}$ that needs not be strongly connected and has no equivalent DPA with ranks $\{1, \dots, k+1\}$. There, the DPA $\mathcal{D}$ has ranks in $\{0, ..., 2m + k + 1\}$, and has a $[2m, 2m + k]$-flower $q_{❀}$. We argue that since $\mathcal{A}$ is strongly connected, $\mathcal{D}$ has only ranks in $\{0, ..., 2m + k\}$.

By [12], if there exists $m \in \mathbb{N}$ and a DPA $\mathcal{D}$ that recognizes $L(\mathcal{A})$ and has a $[2m, 2m + k + 1]$-flower, then $L(\mathcal{A})$ cannot be recognized by a DPA with ranks $\{1, ..., k + 2\}$. Observe that in this case, $L(\mathcal{A})$ cannot be recognized by a DPA with ranks $\{0, ..., k\}$ as well, as by increasing the ranks by 2 we get a DPA with ranks $\{2, ..., k + 2\}$, contradicting the fact $L(\mathcal{A})$ cannot be recognized by a DPA with ranks in $\{1, ..., k + 2\}$. Hence, as $\mathcal{A}$ with ranks $\{0, ..., k\}$ does exist, the DPA $\mathcal{D}$ cannot have a $[2m, 2m + k + 1]$-flower.

Now, in our case, the DPA $\mathcal{A}$, and therefore also $\mathcal{D}$, is strongly-connected. Thus, if $\mathcal{D}$ has a state with rank $2m + k + 1$, then the state $q_{❀}$ is in the same component with this state, and is therefore a $[2m, 2m + k + 1]$ flower. By the above, however, $\mathcal{D}$ cannot have a $[2m, 2m + k + 1]$ flower, implying that $\mathcal{D}$ has ranks in $\{0, ..., 2m + k\}$. □

Let $\mathcal{A}$ and $\mathcal{D}$ be as in Lemma 4.2, and $q_{❀}$ be the $[2m, 2m + k]$-flower in $\mathcal{D}$. Note that $\mathcal{A}$ and $\mathcal{D}$ have the same structure and differ only in their acceptance condition. Let $\Omega = \{0, ..., 2m + k\}$. For a word $w \in \Sigma^*$, let $\rho = s_1, s_2, ..., s_n$ be the run of $\mathcal{D}$ on w. If ρ ends in $q_{❀}$, we define the *$q_{❀}$-loop-abstraction of w* to be the rank-word $\text{abs}(w) \in \Omega^*$ of maximal ranks between successive visits to $q_{❀}$. Formally, let $w = y_0 \cdot y_1 \cdots y_t$ be a partition of w such that $\mathcal{D}$ visits the state $q_{❀}$ after reading the prefix $y_0 \cdots y_j$, for all $0 \leqslant j \leqslant t$, and does not visit $q_{❀}$ in other positions. Then, $\text{abs}(y_i)$, for $0 \leqslant i \leqslant t$, is the maximal rank read along y_i, and $\text{abs}(w) = \text{abs}(y_0) \cdot \text{abs}(y_1) \cdots \text{abs}(y_t)$. Recall that $\mathcal{R}_L = \langle \Sigma, \langle L \rangle, \Delta_L, [\epsilon], \alpha \rangle$, where $\langle L \rangle$ are the equivalence classes of the right-congruence relation on L, thus each state $[u] \in \langle L \rangle$ is associated with the language $u^{-1}L$ of words w such that $uw \in L$. We define a function $\varphi : Q \to \langle L \rangle$ that maps states of $\mathcal{A}$ to languages in $\langle L \rangle$ by $\varphi(q) = L(\mathcal{A}^q)$. Observe that φ is onto. We define a function $\gamma : \langle L \rangle \to Q$ that maps languages in $\langle L \rangle$ to states of $\mathcal{A}$ by arbitrarily choosing for every language $u^{-1}L \in \langle L \rangle$ a state in $\varphi^{-1}(u^{-1}L)$.

We define a sequence of words $u_{2m}, \dots, u_{2m+k} \in \Omega^*$ as follows. The definition proceeds by an induction. Let $M = |Q| + 1$. First, $u_{2m} = (2m)^M$. Then, for $2m < i \leqslant 2m+k$, we have $u_i = (i \cdot u_{i-1})^{M-1} \cdot i$. For example, if $m = 2$ and $|Q| = 2$, then $u_4 = 444$, $u_5 = 544454445$, $u_6 = 6544454445654445444456$, and so on. Let $\mathcal{P}$ be a DFA that accepts a (finite) word $w \in \Sigma^*$ iff the run of $\mathcal{D}$ on w ends in $q_{❀}$ and u_{2m+k} is a suffix of $abs(w)$, for the word $u_{2m+k} \in \Omega^*$ defined above. In Appendix A.7 we describe how to construct $\mathcal{P}$, essentially by combining a DFA over that alphabet Ω that recognizes $\Omega^* \cdot u_{2m+k}$ with a DFA with state space $Q \times \Omega$ that records the highest rank visited between successive visits to $q_{❀}$ and thus abstracts words in Σ^*.

We can now turn to the construction of the DPAs $\mathcal{B}_n$. Recall that $\mathcal{A} = \langle \Sigma, Q, q_0, \Delta, \alpha_{\mathcal{A}} \rangle$, and let $\mathcal{P} = \langle \Sigma, Q_{\mathcal{P}}, t_0, \Delta_{\mathcal{P}}, \{t_{acc}\} \rangle$. For $n \geqslant 1$, we define $\mathcal{B}_n = \langle \Sigma, Q_n, \langle q_0, t_0 \rangle, \Delta_n, \alpha_n \rangle$

as follows. The states of $\mathcal{B}_n$ are $Q_n = (\langle L\rangle \times \{1, \ldots, n\}) \cup (Q \times (Q_{\mathcal{P}} \backslash \{t_{acc}\}))$, where t_{acc} is the unique accepting state of $\mathcal{P}$. We refer to the two components in the union as the *$\mathcal{R}_L$-component* and the *$\mathcal{D}$-component*, respectively. The transitions of $\mathcal{B}_n$ are defined as follows.

- Inside the $\mathcal{R}_L$-component: for every transition $\langle [u], a, [u'] \rangle \in \Delta_L$ and $i \in \{1, \ldots, n-1\}$, there is a transition $\langle ([u], i), a, ([u'], i+1) \rangle \in \Delta_n$.
- From the $\mathcal{R}_L$-component to the $\mathcal{D}$-component: for every transition $\langle [u], a, [u'] \rangle \in \Delta_L$, there is a transition $\langle ([u], n), a, (\gamma([u']), t_0) \rangle \in \Delta_n$.
- Inside the $\mathcal{D}$-component: for every transitions $\langle q, a, q' \rangle \in \Delta$ and $\langle t, a, t' \rangle \in \Delta_{\mathcal{P}}$ with $t' \neq t_{acc}$, there is a transition $\langle (q, t), a, (q', t') \rangle \in \Delta_n$.
- From the $\mathcal{D}$-component to the $\mathcal{R}_L$-component: for every transitions $\langle q, a, q' \rangle \in \Delta$ and $\langle t, a, t_{acc} \rangle \in \Delta_{\mathcal{P}}$, there is a transition $\langle (q, t), a, (\varphi(q'), 1) \rangle \in \Delta_n$.

The acceptance condition of $\mathcal{B}_n$ is induced by that of $\mathcal{A}$. Formally $\alpha_n(q, t) = \alpha_{\mathcal{A}}(q)$, for states $(q, t) \in Q \times Q_{\mathcal{P}}$, and $\alpha_n([u], i) = 0$ for states $([u], i) \in \langle L \rangle \times \{1, \ldots, n\}$.

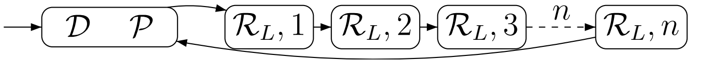


Figure 2: The DPA $\mathcal{B}_n$.

The idea behind the construction of $\mathcal{B}_n$ is as follows. The automaton $\mathcal{B}_n$ stays in $\mathcal{R}_L$ for n steps, then proceeds to a state in $\mathcal{D}$ with the correct residual language, and simulates $\mathcal{D}$ until the ranks corresponding to the word u_{2m+k} have been seen. It then goes back to $\mathcal{R}_L$, by projecting the current state of $\mathcal{D}$ onto its residual in $\langle L \rangle$. The bigger n is, the more time a run spends in the $\mathcal{R}_L$-component, making $\mathcal{R}_L$ the more dominant factor in the sensing cost of $\mathcal{B}_n$. As n tends to infinity, the sensing cost of $\mathcal{B}_n$ tends to that of $\mathcal{R}_L$. The technical challenge is to define $\mathcal{P}$ in such a way that even though the run spends less time in the $\mathcal{D}$ component, we can count on the ranks visited during this short time in order to determine whether the run is accepting. We are now going to formalize this intuition, and we start with the most challenging part of the proof, namely the equivalence of $\mathcal{B}_n$ and $\mathcal{A}$. The proof is decomposed into the three Lemmas 4.4, 4.5, and 4.6.

Lemma 4.4 *Consider a word $u \in \Sigma^*$ such that the run of $\mathcal{B}_n$ on u reaches the $\mathcal{D}$-component in state $\langle q, t \rangle$. Then, $L(\mathcal{D}^q) = L(\mathcal{A}^q) = u^{-1}L$.*

Proof: We prove a stronger claim, namely that if the run of $\mathcal{B}_n$ on u ends in the $\mathcal{R}_L$-component in a state $\langle s, i \rangle$, then $s = [u]$, and if the run ends in the $\mathcal{D}$-component in a state $\langle q, t \rangle$, then $L(\mathcal{A}^q) = u^{-1}L$. The proof proceeds by induction on $|u|$ and is detailed in Appendix A.8. By Lemma 4.2, for every $q \in Q$, we have $L(\mathcal{A}^q) = L(\mathcal{D}^q)$, so the claim follows. □

Lemma 4.5 *If the run of $\mathcal{B}_n$ on a word $w \in \Sigma^\omega$ visits the $\mathcal{R}_L$-component finitely many times, then $w \in L$ iff $w \in L(\mathcal{B}_n)$.*

Proof: Let $u \in \Sigma^*$ be a prefix of w such that the run of $\mathcal{B}_n$ on w stays forever in the $\mathcal{D}$-component after reading u. Let $(q, t) \in Q_n$ be the state reached by $\mathcal{B}_n$ after reading u. By Lemma 4.4, we have $L(\mathcal{A}^q) = u^{-1}L$. Since the run of $\mathcal{B}_n$ from (q, t) stays in the $\mathcal{D}$-components where it simulates the run of $\mathcal{A}$ from q, then $\mathcal{A}^q$ accepts the suffix $w^{|u|}$ iff $\mathcal{B}_n^{(q,t)}$ accepts $w^{|u|}$. It follows that $w \in L$ iff $w \in L(\mathcal{B}_n)$. □

The complicated case is when the run of $\mathcal{B}_n$ on w does visit the $\mathcal{R}_L$-component infinitely many times. This is where the special structure of $\mathcal{P}$ guarantees that the sparse visits in the $\mathcal{D}$-component are sufficient for determining acceptance.

Lemma 4.6 *If the run of $\mathcal{B}_n$ on a word $w \in \Sigma^\omega$ visits the $\mathcal{R}_L$-component infinitely many times, then $w \in L$ iff $w \in L(\mathcal{B}_n)$.*

Proof: Let $\tau = s_1, s_2, s_3, \ldots$ be the run of $\mathcal{B}_n$ on w and let $\rho = q_1, q_2, q_3 \ldots$ be the run of $\mathcal{A}$ on w. We denote by $\tau[i, j]$ the infix $s_i, ..., s_j$ of τ. We also extend $\alpha_{\mathcal{D}}$ to (infixes of) runs by defining $\alpha_{\mathcal{D}}(\tau[i, j]) = \alpha_{\mathcal{D}}(s_i), ..., \alpha_{\mathcal{D}}(s_j)$. For a rank-word $u \in \Omega^*$, we say that an infix $\tau[i, j]$ is a *u-infix* if $\alpha_{\mathcal{D}}(\tau[i, j]) = u$.

If $v = \tau[i, j]$, for some $0 \leqslant i \leqslant j$, is a part of a run of $\mathcal{D}$ that consists of loops around $q_{❀}$, we define the *loop type of* v to be the word in Ω^* that describes the highest rank of each simple loop around $q_{❀}$ in v. An infix of τ whose loop type is u_i for some $2m \leqslant i \leqslant 2m + k$ is called a *u_i-loop-infix.*

By our assumption, τ contains infinitely many u_{2m+k}-infixes. Indeed, by the definition of $\mathcal{P}$, otherwise τ get trapped in the $\mathcal{D}$-component. We proceed by establishing a connection between u_i-loop-infixes of τ and the corresponding infixes of ρ, for all $2m \leqslant i \leqslant 2m + k$.

Let $i \in \{2m, \ldots, 2m + k\}$, and consider a u_i-loop-infix. By the definition of u_i, such a u_i-loop-infix consists of a sequence of $M = |Q| + 1$ i-loops in τ, with loops of lower ranks between them. We can write $w = xvw'$, where $v = w[c, d]$ is the sub word that corresponds to the u_i-loop-infix. Let $u_i' = \alpha_{\mathcal{A}}(\rho[c, d])$ be the ranks of ρ in its part that corresponds to v.

By our choice of M, we can find two indices $c \leqslant j < l \leqslant d$ such that the pairs $\langle(q_j, t), q_j'\rangle$ and $\langle(q_l, t'), q_l'\rangle$ reached by (τ, ρ) in indices j and l, respectively, satisfy $q_j = q_l = q_{❀}$ and $q_j' = q_l'$. Additionally, being a part of the run on a u_i-loop-infix, the highest rank seen between q_j and q_l in τ is i. We write $v = v_1v_2v_3$, where $v_1 = v[1, j]$, $v_2 = v[j + 1, l]$, and $v_3 = v[l + 1, |v|]$. Thus, the loop type of v_2 is in $(iu_{i-1})^+i$, with the convention $u_{2m-1} = \epsilon$.

Consider the runs μ and η of $\mathcal{D}^{q_j}$ and of $\mathcal{A}^{q_j'}$ on v_2^ω, respectively. These runs are loops labeled by v_2, where the highest rank in μ is i. By Lemma 4.4, $L(\mathcal{D}^{q_j}) = L(\mathcal{D}^{q_j'}) = L(\mathcal{A}^{q_j'})$, so the highest rank in η must have same parity (odd or even) as i.

Thus, we showed that for every $i \in \{2m, ..., 2m + k\}$, and for every u_i-loop-infix v of τ, there is an infix of v with loop-type in $(iu_{i-1})^+i$, such that the infix of ρ corresponding to v has highest rank of same parity as i.

We want to show that rank k is witnessed on ρ during every u_{2m+k}-infix of τ. Assume by way of contradiction that this is not the case. This means that there is some u_{2m+k}-infix v' in τ such that all ranks visited in ρ along v' are at most $k - 2$. Indeed, since the highest rank has to be of the same parity as $2m + k$, which has the same parity as k, it cannot be $k - 1$. By the same argument, within v' there is an infix v'' of u_{2m+k-1} of the form $((2m + k - 1)(u_{2m+k-2}))^+(2m + k - 1)$ in which the highest rank in ρ is of the same parity as $k - 1$. As v'' is also an infix of v', the highest rank in ρ along v'' is at most $k - 2$. Thus, the highest rank along v'' is at most $k - 3$. By continuing this argument by induction down to 0, we reach a contradiction (in fact it is reached at level 1), as no rank below 0 is available.

We conclude that the run ρ witnesses a rank k in any u_k-infix of τ. Since τ contains infinitely many u_k-infixes, then ρ contains infinitely many ranks k, and, depending on the parity of k, either both ρ and τ are rejecting or both are accepting.

This concludes the proof that $w \in L$ iff $w \in L(\mathcal{B}_n)$. □

We proceed to show that the sensing cost of the sequence of DPAs $\mathcal{B}_n$ indeed converges to that of $\mathcal{R}_L$.

Lemma 4.7 $\lim_{n\to\infty} scost(\mathcal{B}_n) = scost(\mathcal{R}_L)$.

Proof: Since $\mathcal{D}$ is strongly connected, then $q_{❀}$ is reachable from every state in $\mathcal{D}$. Also, since $q_{❀}$ is a $[2m, 2m + k]$-flower, we can construct a sequence of loops around $q_{❀}$ whose ranks correspond to the word u_{2m+k}. Thus, t_{acc} is reachable from every state in the $\mathcal{D}$-component. This implies that $\mathcal{B}_n$ is strongly connected, and therefore, a run of $\mathcal{B}_n$ is expected to traverse both components infinitely often, making the $\mathcal{R}_L$-component more dominant as n grows, implying that $\lim_{n\to\infty} scost(\mathcal{B}_n) = scost(\mathcal{R}_L)$. Formalizing this intuition involves a careful analysis of $\mathcal{B}_n$'s Markov chain, as detailed in Appendix A.11. □

Lemmas 4.5, and 4.6 put together ensure that for strongly connected automata, we have that $L(\mathcal{B}_n) = L$, so with Lemma 4.7, we get $scost(L) = scost(\mathcal{R}_L)$.

It is left to remove the assumption about $\mathcal{A}$ being strongly connected. The proof is detailed in Appendix A.12, and uses the above result on each ergodic component of $\mathcal{A}$.

Remark 4.8 *All our results can be easily extended to a setting with a non-uniform distribution on the letters given by any Markov chain, or with a different cost for each input in each state. We can also use a decision tree to read the inputs instead of reading them simulatenously, defining for instance a cost of* 1.5 *if the state starts by reading* a*, then if* a *is true it also reads* b*.* □

5 Directions for Future Research

Regular sensing is a basic notion, which we introduced and studied for languages of finite and infinite words. In this section we discuss possible extensions and variants of our definition and contribution.

Open systems: Our setting assumes that all the signals in P are generated by the environment and read by the automaton. In the setting of open systems, we partition P into a set I of input signals, generated by the environment, and a set O of output signals, generated by the system. Then, we define the sensing cost of a specification as the minimal sensing cost required for a transducer that realizes it, where here, sensing is measured only with respect to the signals in I. Also, the transducer does not have to generate all the words in the language – it only has to associate a computation in the language with each input sequence. These two differences may lead to significantly different results than those presented in the paper.

Trade-off between sensing and quality: The key idea in the proof of Theorem 4.1 is that when we reason about languages of infinite words, it is sometimes possible to delay the sensing and only sense in "sparse" intervals. In practice, however, it is often desirable to satisfy eventualities quickly. This is formalized in multi-valued formalisms such as LTL with future discounting [1], where formulas assign higher satisfaction values to computations that satisfy eventualities fast. Our study here suggests that lower sensing leads to lower satisfaction values. An interesting problem is to study and formalize this intuitive trade-off between sensing and quality.

Transient cost: In our definition of sensing, transient states are of no importance. Consequently, for example, all safety languages have sensing cost 0, as the probability of a safety property not being violated is 0, and once it is violated, no sensing is required. An alternative definition of sensing cost may take transient states into an account. One way to do it is to define the sensing cost of a run as the discounted sum $\sum_{i\geqslant 0} 2^{-i} \cdot sensed(|q_i|)$ of the sensing costs of the states $q_0, q_1, ...$ it visits.

Beyond regular: Our definition of sensing cost can be adapted to more complex models, such as pushdown automata or Turing machines. It would be interesting to see the trade-off between sensing and classical complexity measures in such models.

References

[1] S. Almagor, U. Boker, and O. Kupferman. Discounting in LTL. In *Proc. 20th TACAS*, LNCS 8413, pages 424-439. Springer, 2014.

[2] G. Avni and O. Kupferman. When does abstraction help? *Information Processing Letters*, 113:901–905, 2013.

[3] K. Chatterjee and R. Majumdar. Minimum attention controller synthesis for ω-regular objectives. In *FORMATS*, pages 145–159, 2011.

[4] K. Chatterjee, R. Majumdar, and T. A. Henzinger. Controller synthesis with budget constraints. In *Proc. 11th HSCC*, LNCS 4981, pages 72–86. Springer, 2008.

[5] D.L. Donoho. Compressed sensing. *IEEE Trans. Inform. Theory*, 52:1289–1306, 2006.

[6] C. Grinstead and J. Laurie Snell. 11:markov chains. In *Introduction to Probability*. American Mathematical Society, 1997.

[7] G. Kindler. *Property Testing, PCP, and Juntas*. PhD thesis, Tel Aviv University, 2002.

[8] O. Kupferman and M.Y. Vardi. Church's problem revisited. *The Bulletin of Symbolic Logic*, 5(2):245 – 263, 1999.

[9] E. Kushilevitz and N. Nisan. *Communication complexity*. Cambridge Univ. Press, 1997.

[10] C. Mauduit and A. Sárköz. On finite pseudorandom binary sequences. i. measure of pseudorandomness, the legendre symbol. *Acta Arith.*, 82(4):365–377, 1997.

[11] S. Muthukrishnan. Theory of data stream computing: where to go. In *Proc. 30th PODS*, pages 317–319, 2011.

[12] D. Niwinski and I. Walukiewicz. Relating hierarchies of word and tree automata. In *STACS*, LNCS 1373. Springer, 1998.

[13] S. Schewe. Beyond Hyper-Minimisation—Minimising DBAs and DPAs is NP-Complete. In *Proc. 30th FSTTCS*, LIPIcs 8, pages 400–411, 2010.

A Details and Proofs

A.1 Proof of Lemma 2.2

Let $q \in Q$, and consider a random infinite run r in M_A. If q is transient, then it is easy to see that $\lim_{m\to\infty} \frac{1}{m} E_m(q) = 0 = \pi(q)$, because with probability 1, the state q does not appear after some point in r. Otherwise, let $C \in \mathcal{C}_e$ be the ergodic SCC of q. The probability that r reaches C is given by $\rho(C)$. By the law of total expectation, and since q is reachable only if r reaches C, we have that $E_m(q) = \rho(C) E^C_{m-t}$ where t is the expected time by which r reaches C. Thus, $\lim_{m\to\infty} \frac{E_m(q)}{m} = \rho(C) \lim_{m\to\infty} \frac{E^C_{m-t}}{m} = \rho(C) \lim_{m\to\infty} \frac{E^C_m}{m} = \rho(C)\pi_C(q)$. □

A.2 Proof of Theorem 2.3

By Lemma 2.2, we have $\pi(q) = \lim_{m\to\infty} \frac{E_m(q)}{m}$, where $E_m(q)$ is the expected number of occurrences of q in a random m-step run. This can be restated in our case as $\pi(q) = \lim_{m\to\infty} \frac{1}{m|\Sigma|^m} \sum_{w:|w|=m} Occ_w(q)$, where $Occ_w(q)$ is the number of occurrences of q in the run of $\mathcal{A}$ on w. By definition, $scost(\mathcal{A}) = \lim_{m\to\infty} |\Sigma|^{-m} \sum_{w:|w|=m} scost_{\mathcal{A}}(w)$, and also $scost_{\mathcal{A}}(w) = \sum_{q\in Q} scost(q) \cdot Occ_w(q)$. From this, we get

$$\begin{aligned} scost(\mathcal{A}) &= \lim_{m\to\infty} |\Sigma|^{-m} \sum_{w:|w|=m} \sum_{q\in Q} scost(q) \cdot Occ_w(q) \\ &= \sum_{q\in Q} scost(q) \cdot \lim_{m\to\infty} |\Sigma|^{-m} \sum_{w:|w|=m} Occ_w(q) = \sum_{q\in Q} scost(q) \cdot \pi(q). \end{aligned}$$

□

A.3 Calculating the SCC-reachability distribution

The stationary distribution π_C of each ergodic SCC C can be computed in polynomial time by solving a system of linear equations.

We show that the SCC-reachability distribution $\rho : \mathcal{C} \to [0,1]$ can also be calculated in polynomial time. First, if the initial state q_0 is in an ergodic SCC, the result is trivial. Otherwise, we proceed as follows. We associate with $\mathcal{A}$ the Markov chain M'_A, in which we contract each ergodic SCC of $\mathcal{A}$ to a single state. That is, M'_A is obtained from M_A by replacing each $C \in \mathcal{C}_e$ by a single state q_C. Notice that M'_A is an *absorbing* Markov chain, thus it reaches a sink state with probability 1. Indeed, the probability of reaching an ergodic SCC in M_A is 1, and every SCC in M_A becomes a sink state in M'_A.

By indexing the rows and columns in the transition matrix of M'_A such that transient states come before ergodic states, we can put the matrix in a normal form $\begin{pmatrix} T & E \\ 0 & I \end{pmatrix}$, where T describes the transitions between transient states, E from transient to ergodic states, and I is the identity matrix of size $|\mathcal{C}_e|$. Note that, indeed, there are no transitions from ergodic states to transient ones, which explains the 0 matrix in the bottom left, and that I captures the fact the ergodic states are sinks. By [6], the entry at coordinates (q_t, q_C) in the matrix $B = (I - T)^{-1}E$ is the probability of reaching the sink q_C starting from the transient state q_t. Therefore, for every $C \in \mathcal{C}_e$, we have that $\rho(C) = B_{(q_0,q_C)}$.

A.4 Examples of sensing costs of automata

Let $P = \{a, b\}$. Consider the DFA $\mathcal{A}_1$ appearing in Figure 3. Note that $L(\mathcal{A}_1) = (\texttt{True})^* \cdot a \cdot b \cdot (\texttt{True})^*$. It is easy to see that $sensed(q_0) = \{a\}$, $sensed(q_1) = \{b\}$, and $sensed(q_2) = \emptyset$. Accordingly, $scost(q_0) = scost(q_1) = 1$ and $scost(q_2) = 0$. Since the state q_2 forms the only ergodic SCC, the limiting distribution on the states of $\mathcal{A}$ is $\pi(q_0) = \pi(q_1) = 0$ and $\pi(q_2) = 1$. Hence, $scost(\mathcal{A}_1) = 0$.

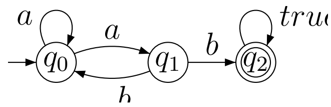


Figure 3: The DFA $\mathcal{A}_1$.

Consider now the DFA $\mathcal{A}_2$, appearing in Figure 4, with $L(\mathcal{A}_2) = (\texttt{True})^* \cdot a \cdot b$. Here, $sensed(q_0) = \{a\}$, $sensed(q_1) = \{a, b\}$, and $sensed(q_2) = \{a\}$. Accordingly, $scost(q_0) = scost(q_2) = 1$ and $scost(q_2) = 2$. Since $\mathcal{A}_2$ is strongly connected, its limiting distribution is its unique stationary distribution, which can be calculated by solving the following system of equations, where x_i corresponds to $\pi(q_i)$:

- $x_0 = \frac{1}{2}x_0 + \frac{1}{4}x_1 + \frac{1}{2}x_2$. ﬀl $x_2 = \frac{1}{2}x_1$.
- $x_1 = \frac{1}{2}x_0 + \frac{1}{4}x_1 + \frac{1}{2}x_2$. ﬀl $x_0 + x_1 + x_2 = 1$.

Accordingly, $\pi(q_0) = \pi(q_1) = \frac{2}{5}$ and $\pi(q_2) = \frac{1}{5}$. We conclude that the sensing cost of $\mathcal{A}_2$ is $1 \cdot \frac{2}{5} + 2 \cdot \frac{2}{5} + 1 \cdot \frac{1}{5} = \frac{7}{5}$.

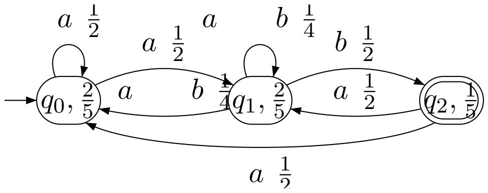


Figure 4: The DFA $\mathcal{A}_2$ and its corresponding Markov chain.

A.5 Proof that the cost of the residual automaton is necessary

We prove that for every DPA $\mathcal{A}$ with $L(\mathcal{A}) = L$, we have that $scost(\mathcal{A}) \geqslant scost(\mathcal{R}_L)$. Consider a word $w \in \Sigma^\omega$ and a prefix $u \in \Sigma^*$ of w. After reading u, the DPA $\mathcal{R}_L$ reaches the state $[u]$ and the DPA $\mathcal{A}$ reaches a state q with $L(\mathcal{A}^q) = u^{-1}L$. As in the case of finite words, for every state $q \in Q$ such that $L(\mathcal{A}^q) = u^{-1}L$, it holds that $sensed([u]) \subseteq sensed(q)$, implying that $scost(q) \geqslant scost([u])$. Now, since this holds for all prefixes u of w, it follows that $scost_{\mathcal{A}}(w) \geqslant scost_{\mathcal{R}_L}(w)$. Finally, since the latter holds for every word $w \in \Sigma^\omega$, it follows that $scost(\mathcal{A}) \geqslant scost(\mathcal{R}_L)$.

Note that the arguments in the proof are independent of the acceptance condition of $\mathcal{A}$ and apply also to stronger acceptance conditions, such as the Muller acceptance condition.

A.6 Proof of Claim 4.3

Assume by contradiction that there is a DPA $\mathcal{B}$ with ranks $\{1, \ldots, k+1\}$ that recognizes $L(\mathcal{A})$. Let $u \in \Sigma^*$ be such that the run of $\mathcal{B}$ on u ends in an ergodic SCC C of $\mathcal{B}$. Since $\mathcal{A}$ is strongly connected, there is a word $v \in \Sigma^*$ such that the run of $\mathcal{A}$ on the word uv ends in the initial state of $\mathcal{A}$. That is, $(uv)^{-1}L = L$. Since $L(\mathcal{B}) = L(\mathcal{A})$, this implies that the run of $\mathcal{B}$ on uv ends in a state q such that $L(\mathcal{B}^q) = L$. Since after reading u, the run is in the ergodic component C, we have $q \in C$. Thus, $\mathcal{B}^q$ is a strongly connected DPA equivalent to $\mathcal{A}$. By our assumption, $L(\mathcal{A})$ cannot be recognized by a DPA with ranks in $\{1, ..., k\}$, and thus the ranks in $\mathcal{B}^q$ are $\{1, ..., k+1\}$. In particular, there is a state q' in $\mathcal{B}^q$ with rank $k+1$.

Let s be a state in $\mathcal{A}$ with rank k. Let $u_0 \in \Sigma^*$ be such that the run of $\mathcal{A}$ on u_0 reaches s, and let $u_1 \in \Sigma^*$ be such that the run of $\mathcal{B}^q$ on $u_0 u_1$ reaches q' (recall that $\mathcal{B}^q$ is strongly connected). We can construct in this manner an infinite word $w = u_0 u_1 u_2 \cdots$ such that for all $i \geqslant 0$, the run of $\mathcal{A}$ on $u_0, ..., u_{2i}$ reaches s and the run of $\mathcal{B}^q$ on $u_0, ..., u_{2i+1}$ reaches q'. Thus, the maximal rank in the run of $\mathcal{A}$ (resp. $\mathcal{B}^q$) on w is k (resp. $k+1$). However, k and $k+1$ have different parity, so $L(\mathcal{A}) \neq L(\mathcal{B}^q)$, which contradicts our assumption. We conclude that $L(\mathcal{A})$ is not recognizable by a DPA with ranks $\{1, ..., k+1\}$.

A.7 The construction of the auxiliary DFA $\mathcal{P}$

Let $\mathcal{H}_{2m+k} = \langle \Omega, Q', q'_0, \Delta', \alpha' \rangle$ be the minimal DFA that recognizes the language $\Omega^* \cdot u_{2m+k}$. We can define $\mathcal{H}_{2m+k}$ so that α' contains a single state q'_{acc}. Indeed, there is a single accepting Myhill-Nerode class of the language $\Omega^* \cdot u_{2m+k}$.

Let $\mathcal{H}$ be the DFA with state space $Q \times \Omega$ and alphabet Σ that maintains in its state the highest rank seen since the last occurrence of $q_{❀}$ (or since the beginning of the word, if no $q_{❀}$ has been seen) in the run of $\mathcal{D}$ on the word. Thus, $\mathcal{H}$ is in state $\langle q, i \rangle$ iff the highest rank that was visited by $\mathcal{D}$ since the last visit to $q_{❀}$ is i. Observe that simulating $\mathcal{H}$ when $\mathcal{D}$ is in an r-loop that started from $q_{❀}$, means that the next visit to $q_{❀}$ will make $\mathcal{H}$ reach the state $\langle q_{❀}, r \rangle$.

Formally, $\mathcal{H} = \langle \Sigma, Q \times \Omega, \langle q_0, 0 \rangle, \Delta_{\mathcal{H}}, Q \times \Omega \rangle$, where $\Delta_{\mathcal{H}}$ is defined as follows.

- For every state $\langle q, i \rangle$ where $q \neq q_{❀}$, and for every $\sigma \in \Sigma$, we have $\langle \langle q, i \rangle, \sigma, \langle s, \max\{i, i'\} \rangle \rangle \in \Delta_{\mathcal{H}}$ where s is such that $\langle q, \sigma, s \rangle \in \Delta$, and $i' = \alpha_{\mathcal{D}}(s)$.

- For a state $\langle q_{❀}, i \rangle$ and for $\sigma \in \Sigma$, we have $\langle \langle q_{❀}, i \rangle, \sigma, \langle s, i' \rangle \rangle \in \Delta_{\mathcal{H}}$ where s is such that $\langle q, \sigma, s \rangle \in \Delta$, and $i' = \alpha_{\mathcal{D}}(s)$.

We obtain $\mathcal{P}$ by composing $\mathcal{H}$ with $\mathcal{H}_{2m+k}$ as follows. In every step of a run of $\mathcal{D}$, the DFA $\mathcal{P}$ advances in the DFA $\mathcal{H}$, while the DFA $\mathcal{H}_{2m+k}$ only advances when we visit $q_{❀}$, and it advances according to the highest rank stored in $\mathcal{H}$.

Formally, $\mathcal{P} = \langle \Sigma, Q_{\mathcal{P}}, t_0, \Delta_{\mathcal{P}}, \{t_{acc}\} \rangle$, where $Q_{\mathcal{P}} = Q \times \Omega \times Q'$, $t_0 = \langle q_0, 0, q'_0 \rangle$, $t_{acc} = \langle q_{❀}, 2m + k, q'_{acc} \rangle$ and the transition relation is defined as follows. For every state $\langle q, i, s \rangle \in q_{\mathcal{P}}$ and letter $\sigma \in \Sigma$, we have $\langle \langle q, i, s \rangle, \sigma, \langle q', i', s' \rangle \rangle \in \Delta_{\mathcal{P}}$, where $\langle q', i' \rangle$ is such that $\langle \langle q, i \rangle, \sigma, \langle q', i' \rangle \rangle \in \Delta_{\mathcal{H}}$, and s' is such that $\langle s, i', s' \rangle \in \Delta'$ if $q' = q_{❀}$, while $s' = s$ if $q \neq q_{❀}$.

A.8 Details of the proof of Lemma 4.4

We complete the proof by induction.

For $u = \epsilon$, the claim is trivial, as $\mathcal{B}_n$ starts in $\langle q_0, t_0 \rangle$. Consider the word $u \cdot \sigma$ for $u \in \Sigma^*$ and $\sigma \in \Sigma$. By the induction hypothesis, if the run of $\mathcal{B}_n$ on u ends in an $\mathcal{R}_L$ component in state $\langle s, i \rangle$, then $s = [u]$. If $i < n$, then, by the definition of $\mathcal{R}_L$, the next state is $\langle [u \cdot \sigma], i + 1 \rangle$, we are done. If $i = n$ then the next state is $\langle \gamma([u \cdot \sigma]), t_0 \rangle$. By the definition of γ, we have $L(\mathcal{A}^{\gamma([u \cdot \sigma])}) = (u \cdot \sigma)^{-1} L$, so we are done.

We continue to the case the run of $\mathcal{B}_n$ on u ends in the $\mathcal{D}$-component. If the run ends in a state $\langle p, t \rangle$ such that $\langle t, \sigma, t_{acc} \rangle \notin \Delta_{\mathcal{P}}$, then, by the induction hypothesis, we have that $L(\mathcal{A}^p) = u^{-1} L$. Reading σ, we move to a state $\langle p', t' \rangle$ such that $\langle p, \sigma, p' \rangle \in \Delta$, thus $L(\mathcal{A}^{p'}) = (u \cdot \sigma)^{-1} L$, and we are done. Otherwise, $\langle t, \sigma, t_{acc} \rangle \in \Delta_{\mathcal{P}}$ and the next state of $\mathcal{B}_n$ is $\langle \varphi(p'), 1 \rangle$. By the definition of φ, we have $\varphi(p') = [u \cdot \sigma]$, and we are done.

A.9 On Monotonicity of Sensing

The example in Remark 2.1 suggests that there is a trade-off between guessing and sensing. Consider a DFA $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, \alpha \rangle$, with $\Sigma = 2^P$. For a state $q \in Q$ and a signal $p \in P$, let $\mathcal{A}_{q \downarrow p}$ be the NFA obtained from $\mathcal{A}$ by ignoring p in q. Thus, in state q, the NFA $\mathcal{A}_{q \downarrow p}$ guesses the value of p and proceeds to all the successors that are reachable with some value. Note that the guess introduces nondeterminism. Formally, $\mathcal{A}_{q \downarrow p} = \langle \Sigma, 2^Q, \{q_0\}, \delta', \alpha' \rangle$, where for every state $T \in 2^Q$ and letter $S \in 2^P$, we define $\delta'(T, S) = \bigcup_{t \in T} \delta(t, S)$ if $q \notin T$, and $\delta'(T, S) = \delta(q, S \backslash \{p\}) \cup \delta(q, S \cup \{p\}) \cup \bigcup_{t \in T \backslash \{q\}} \delta(t, S)$ if $q \in T$. Also, a state $T \subseteq Q$ is in α' iff $T \cap \alpha \neq \emptyset$. It is easy to see that $L(\mathcal{A}) \subseteq L(\mathcal{A}_{q \downarrow p})$. Since $\mathcal{A}_{q \downarrow p}$ is obtained from $\mathcal{A}$ by giving up some of its sensing, it is tempting to think that $scost(L(\mathcal{A}_{q \downarrow p})) \leqslant scost(L(\mathcal{A}))$. As we now show, however, sensing is not monotone. For two languages L and L', we say that L' is an abstraction of L if there is a DFA $\mathcal{A}$ such that $L(\mathcal{A}) = L$ and there is a state q and a signal p of $\mathcal{A}$ such that $L' = L(\mathcal{A}_{q \downarrow p})$.

Theorem A.1 *Sensing is not monotone. That is, there is a language L and an abstraction L' of L such that $scost(L) \leqslant scost(L')$.*

Proof: Let $P = \{a, b, c\}$. Consider the language $L = a \cdot \texttt{True}^* \cdot b + (\neg a) \cdot \texttt{True}^* \cdot c$. It is not hard to see that $scost(L) = 1$. Indeed, a DFA for L has to sense a in its initial state and then has to always sense either b (in case a appears in the first letter) or c (otherwise).

Giving up the sensing of a in the initial state of a DFA for L we end up with the language $L' = (\texttt{True})^{+} \cdot (b \vee c)$. It is not hard to see that $scost(L') = 2$. Indeed, every DFA for L' has to almost always sense both b and c. $\square$

We conclude that replacing a sensor with non-determinism may actually result in a language for which we need more sensors. This corresponds to the known fact that abstraction of automata may result in bigger (in fact, exponentially bigger) DFAs [2]. Also, while the above assumes an abstraction that over-approximates the original language, a dual argument could show that under-approximating the language (that is, defining $\mathcal{A}_{q\downarrow p}$ as a universal automaton) may result in a language with higher sensing cost.

A.10 Operations on DFAs and Their Sensing Cost

In this section we study the effect of actions on DFAs on their sensing cost. We start with complementation. For every regular language L, a DFA for $comp(L) = \Sigma^{*} \backslash L$ can be obtained from a DFA for L by complementing the set of accepting states. In particular, this holds for $\mathcal{R}_L$, implying the following.

Lemma A.2 *For every regular language* L *we have that* $scost(L) = scost(comp(L))$.

Next, we consider the union of two regular languages.

Lemma A.3 *For every regular languages* $L_1, L_2 \subseteq (2^P)^{*}$, *we have* $scost(L_1 \cup L_2) \leqslant scost(L_1) + scost(L_2)$.

Proof: Consider the minimal DFAs $\mathcal{A}_1 = \langle 2^P, Q^1, \delta^1, q_0^1, \alpha^1 \rangle$ and $\mathcal{A}_2 = \langle 2^P, Q^2, \delta^2, q_0^2, \alpha^2 \rangle$ for L_1 and L_2, respectively. Let $\mathcal{B} = \langle 2^P, Q^1 \times Q^2, \delta, (q_0^1, q_0^2), (\alpha^1 \times Q^2) \cup (Q^1 \times \alpha^2) \rangle$ be their product DFA. Note that $L(\mathcal{B}) = L_1 \cup L_2$. We claim that for every state $\langle q, s \rangle \in Q^1 \times Q^2$, we have that $sensed(\langle q, s \rangle) \subseteq sensed(q) \cup sensed(s)$. Indeed, if $p \notin sensed(q) \cup sensed(s)$, then for every set $S \subseteq P \backslash \{p\}$, it holds that $\delta^1(q, S) = \delta^1(q, S \cup \{p\})$ and $\delta^2(s, S) = \delta^2(s, S \cup \{p\})$. Thus, $\delta(\langle q, s \rangle, S) = \delta(\langle q, s \rangle, S \cup \{p\})$, so $p \notin sensed\langle q, s \rangle$. We thus have that $scost(\langle q, s \rangle) \leqslant scost(q) + scost(s)$.

It follows that for every word $w \in (2^P)^{*}$, we have that $scost_{\mathcal{B}}(w) \leqslant scost_{\mathcal{A}_1}(w) + scost_{\mathcal{A}_2}(w)$. Indeed, in every state in the run of $\mathcal{B}$ on w, the sensing is at most the sum of the sensings in the corresponding states in the runs of $\mathcal{A}_1$ and $\mathcal{A}_2$ on w. Since this is true for every word in Σ^{*}, then taking the limit of the average cost yields the result. $\square$

We now consider the concatenation of two languages. The following lemma shows that the sensing level may increase from 0 to 1 when concatenating languages.

Lemma A.4 *There are languages* $L_1, L_2 \subseteq \Sigma^{*}$ *such that* $scost(L_1) = scost(L_2) = 0$, *yet* $scost(L_1 \cdot L_2) = 1$.

Proof: Let $P = \{a\}$, and consider the languages $L_1 = (2^P)^*$ and $L_2 = \{\{a\}\}$. It is not hard to see that $scost(L_1) = scost(L_2) = 0$. Indeed, a DFA for L_1 consists of a single accepting sink with no sensing, and a DFA for L_2 has a single ergodic component, which is a rejecting sink with no sensing. On the other hand $L_1 \cdot L_2$ consists of all words that end with $\{a\}$ and thus a DFA for it has to always sense a. □

A.11 Formal proof of Lemma 4.7

Consider the Markov chain that corresponds to $\mathcal{B}_n$, and let T_n be its transition matrix. For a vector $v = (v_1, \ldots, v_m)$, let $\|v\| = \sum_{i=1}^{m} v_i$. The sensing cost of $\mathcal{B}_n$ is computed using the limiting distribution π_n of $\mathcal{B}_n$. Since $\mathcal{B}_n$ is strongly connected, it has a unique stationary distribution. Thus π_n is obtained as a solution of the equation $\pi_n T_n = \pi_n$, subject to the constraint $\|\pi_n\| = 1$. We denote by $x_n = (x_{n,1}, \ldots, x_{n,d})$ the sub-vector of π_n that corresponds to the $\mathcal{D}$-component, and denote by $y_{n,i}$ the sub-vector that corresponds to the i-th $\mathcal{R}_L$-component. For every $1 \leqslant i < n$, it is easy to see that $\|y_{n,i}\| = \|y_{n,i+1}\|$. Indeed, all the transitions from the i-th copy of $\mathcal{R}_L$ are to the $(i+1)$-th copy. Thus, $\|y_{n,i}\|$ is independent of i. Let $a_n = \|y_{n,1}\| \geqslant 0$ and $b_n = \|x_n\| \geqslant 0$. Observe that for every n, we have that $na_n + b_n = 1$, so in particular, $\lim_{n\to\infty} a_n = 0$.

Let $\epsilon > 0$. By the definition of $\mathcal{P}$, we always enter the first $\mathcal{R}_L$-component in the state $[q_{❀}]$ of $\mathcal{R}_L$ – the state corresponding to $L(\mathcal{A}^{q_{❀}})$. Let τ_0 be the distribution over the states of $\mathcal{R}_L$ in which $[q_{❀}]$ is assigned probability 1 and the other states of $\mathcal{R}_L$ are assigned 0, and let $\theta = (\theta_1, \ldots, \theta_l)$ be the unique stationary distribution of $\mathcal{R}_L$. Let R be the matrix associated with the Markov chain of $\mathcal{R}_L$, and let $\tau_i = \tau_0 R^i$ for every $i \geqslant 1$. By [6], there exists n_0 such that for all index $i \geqslant n_0$ and $1 \leqslant j \leqslant l$, we have that $|\tau_{i,j} - \theta_j| \leqslant \epsilon$. Note that for all n and i, it holds that $y_{n,i} = \tau_i$.

Let $\{q_1, \ldots, q_d\}$ be the states in the $\mathcal{D}$-component. Since $\mathcal{P}$ is strongly connected, then for every $1 \leqslant i, j \leqslant d$ there is a path from q_i to q_j with at most $d-1$ transitions. Since there are at most $|\Sigma|$ edges leaving each state, the probability of taking each edge along such a path is at least $\mu = \frac{1}{|\Sigma|}$. Therefore, the probability of reaching q_j from q_i is at least μ^{d-1}. Consider the maximal entry in x_n (w.l.o.g $x_{n,1}$). It holds that $x_{n,1} \geqslant \frac{\|x_n\|}{d} = \frac{b_n}{d}$. Therefore, for all $1 \leqslant j \leqslant d$, we have $x_{n,j} \geqslant \mu^{d-1} x_{n,1} \geqslant \frac{\mu^{d-1}}{d} b_n$.

Recall that t_{acc} is reachable from all the states in the $\mathcal{D}$-component. Therefore, there is at least one transition from some state q_j of the $\mathcal{D}$-component to the first $\mathcal{R}_L$-component. This means that $a_n \geqslant \mu \cdot x_{n,j} \geqslant \frac{\mu^d}{d} b_n$, implying that $b_n \leqslant \mu^{-d} \cdot d \cdot a_n$, which tends to 0 when n tends to ∞.

We now consider the cost of $\mathcal{B}_n$, for $n \geqslant n_0$. Clearly, the maximal cost of a state is $|P|$. Let c_j be the cost of the state indexed j in $\mathcal{R}_L$, and let $\tau_i = (\tau_{i,1} \ldots, \tau_{i,l})$. Then,

$$\begin{aligned} scost(\mathcal{B}_n) &\leqslant b_n|P| + n_0 a_n |P| + a_n \textstyle\sum_{i=n_0}^{n} \sum_{j=1}^{d} \tau_{i,j} c_j \\ &\leqslant b_n|P| + n_0 a_n |P| + a_n \textstyle\sum_{i=n_0}^{n} \sum_{j=1}^{d} (\theta_j + \epsilon) c_j. \end{aligned}$$

Therefore, when $n \to \infty$, as $a_n \to 0$ and $b_n \to 0$, we get $scost(\mathcal{B}_n) \leqslant (n - n_0)a_n \sum_{j=1}^{d} \theta_j c_j + O(\epsilon) + o(1)$. But we know $na_n + b_n = 1$, and $b_n \to 0$, so $na_n \to 1$, and therefore $(n - n_0)a_n \to 1$. We get $scost(\mathcal{B}_n) \leqslant scost(\mathcal{R}_L) + O(\epsilon) + o(1)$. Furthermore, by Lemmas 4.5 and 4.6, for all n we have $L(\mathcal{B}_n) = L(\mathcal{A})$, thus $scost(\mathcal{R}_L) \leqslant scost(\mathcal{B}_n)$.

Since the above holds for all $\epsilon > 0$, we conclude that $\lim_{n\to\infty} scost(\mathcal{B}_n) = scost(\mathcal{R}_L)$.

A.12 Proof of the non-SCC case of Theorem 4.1

Assume then that $\mathcal{A}$ is not a strongly connected DPA, and let $C_1, \ldots, C_l$ be its ergodic SCCs. For each $1 \leqslant i \leqslant l$ and $q \in C_i$, let L_i^q the language recognizing by C_i, with q as initial states. Remark that the residual automata R_i^q of languages L_i^q only differ in their initial states. We call R_i the common automaton where no initial state is defined. For each $i \in [1, l]$ and $q \in C_i$, we have $scost(L_i^q) = scost(R_i)$.

We can now perform the above construction on R_i, which will work simultaneously for all initial states. This yields automata $(\mathcal{B}_{1,n}, \ldots, \mathcal{B}_{l,n})_{n\geqslant 1}$ with no initial state specified, such that for each $i \in [1, l]$:

1. $\lim_{n\to\infty} scost(B_{i,n}) = scost(R_i)$
2. For all $q \in C_i$ and $n \geqslant 1$, there is a state q_n in $B_{i,n}$ such that $B_{i,n}$ with q_n as initial state recognizes exactly L_i^q.

Let $\mathcal{A}_n$ be the DPA obtained from $\mathcal{A}$ by replacing each SCC C_i by $\mathcal{B}_{i,n}$, with the entry points to $\mathcal{B}_{i,n}$ being chosen to preserve the correct residual language. More formally, if a transition (p, a, q) of $\mathcal{A}$ will be replaced by (p, a, q_n) in $\mathcal{A}_n$, where q_n is as defined in the second item above.

This construction ensures that for all $n \geqslant 1$, we have $L(\mathcal{A}_n) = L(\mathcal{A})$. Indeed, if a word entered a component C_i in $\mathcal{A}$, it will now enter a component $\mathcal{B}_{i,n}$ in $\mathcal{A}_n$, in a state q_n that matches the correct residual language. Then, the correctness of the construction in the SCC case guarantees that the word is accepted if and only if it is in L.

It remains to show that $\lim_{n\to\infty} scost(\mathcal{A}_n) \to scost(\mathcal{R}_L)$, therefore witnessing the wanted result: $scost(L) = scost(\mathcal{R}_L)$.

Let ρ (resp. ρ_n) be the SCC-reachability distribution of $\mathcal{A}$ (resp. $\mathcal{A}_n$). Recall that the ergodic components $(C_i)_{1\leqslant i\leqslant l}$ in $\mathcal{A}$ have been replaced by $(\mathcal{B}_{i,n})_{1\leqslant i\leqslant l}$ in $\mathcal{A}_n$, and the transient component have been left unchanged. Thefore, for every $1 \leqslant i \leqslant l$ and $n \geqslant 1$, we have that $\rho(C_i) = \rho_n(\mathcal{B}_{i,n})$. By Theorem 2.3, we obtain $scost(\mathcal{A}_n) = \sum_{i=1}^{l} \rho(C_i) scost(\mathcal{B}_{i,n})$. When n tends to ∞, we get $\sum_{i=1}^{l} \rho(C_i) scost(R_i)$ (by item 1 above).

Finally, let $\mathcal{A}_R$ be the DPA obtained from $\mathcal{A}$ by replacing each SCC C_i by its residual automaton R_i, again keeping the entry points to R_i consistent with residuals (here there is no choice: the states of R_i are exactly the possible residuals).

Since the SCC-reachability distribution in $\mathcal{A}$ and $\mathcal{A}_R$ coincide, it follows that $scost(\mathcal{A}_R) = \sum_{i=1}^{l} \rho(C_i) scost(R_i) = \lim_{n\to\infty} scost(\mathcal{A}_n)$. It remains to show that $\mathcal{A}_R$ has same cost as the residual automaton $\mathcal{R}_L$ of L, and we can conclude $\lim_{n\to\infty} scost(\mathcal{A}_n) = scost(\mathcal{R}_L)$, and finally $scost(L) \leqslant scost(\mathcal{R}_L)$. Since the opposite inequality is always true by Appendix A.5, we get $scost(L) = scost(\mathcal{R}_L)$

Lemma A.5 $scost(\mathcal{A}_R) = scost(\mathcal{R}_L)$.

Proof: Let $D_1, \dots, D_k$ be the ergodic SCCs of $\mathcal{R}_L$. For each $i \in [1, l]$, and $q \in C_I$, there must be j_i^q such that L_i^q is a state of $D_{j_i^q}$. Moreover, j_i^q does not depend on q, since both the C_i and the D_j are strongly connected. Therefore, each C_i can be mapped to some D_{j_i} such the states of D_{j_i} are exactly the L_i^q for $q \in C_i$. Actually for each i, the automata R_i and D_{j_i} are exactly the same, if initial states are omitted.

Let ρ be the SCC-reachability distribution of $\mathcal{A}$ (or equivalently $\mathcal{A}_R$) and σ the one of $\mathcal{R}_L$. Because the residual languages have to match in $\mathcal{A}$ and $\mathcal{R}_L$, for each $j \in [1, k]$, we have $\sigma(D_j) = \sum_{j_i=j} \rho(C_i)$. Therefore, $scost(\mathcal{R}_L) = \sum_{j=1}^{k} \sigma(D_j) scost(D_j) = \sum_{j=1}^{k} \sum_{j_i=j} \rho(C_i) scost(D_j) = \sum_{i=1}^{l} \rho(C_i) scost(R_i) = scost(\mathcal{A}_R)$. □

The Sensing Cost of Monitoring and Synthesis*

Shaull Almagor[†] *Denis Kuperberg* [‡] *Orna Kupferman* [§]

Abstract

In [2], we introduced *sensing* as a new complexity measure for the complexity of regular languages. Intuitively, the sensing cost quantifies the detail in which a random input word has to be read by a deterministic automaton in order to decide its membership in the language. In this paper, we consider sensing in two principal applications of deterministic automata. The first is *monitoring*: we are given a computation in an on-line manner, and we have to decide whether it satisfies the specification. The second is *synthesis*: we are given a sequence of inputs in an on-line manner and we have to generate a sequence of outputs so that the resulting computation satisfies the specification. In the first, our goal is to design a monitor that handles all computations and minimizes the expected average number of sensors used in the monitoring process. In the second, our goal is to design a transducer that realizes the specification for all input sequences and minimizes the expected average number of sensors used for reading the inputs.

We argue that the two applications require new and different frameworks for reasoning about sensing, and develop such frameworks. We focus on safety languages. We show that for monitoring, minimal sensing is attained by a monitor based on the minimal deterministic automaton for the language. For synthesis, however, the setting is more challenging: minimizing the sensing may require exponentially bigger transducers, and the problem of synthesizing a minimally-sensing transducer is EXPTIME-complete even for safety specifications given by deterministic automata.

[*] Published in the proceedings of the 35th International Conference on Foundation of Software Technology and Theoretical Computer Science, LIPIcs 45, pages 380–393, Schloss Dagstuhl, 2015.

[†] The Hebrew University, Jerusalem, Israel.

[‡] The University of Warsaw, Faculty of Mathematics, Informatics, and Mechanics.

[§] The Hebrew University, Jerusalem, Israel.

1 Introduction

Studying the complexity of a formal language, there are several complexity measures to consider. When the language is given by means of a Turing Machine, the traditional measures are time and space requirements. Theoretical interest as well as practical considerations have motivated additional measures, such as randomness (the number of random bits required for the execution) [11] or communication complexity (number and length of messages required) [10]. For ω-regular languages, given by means of finite-state automata, the classical complexity measure is the size of a minimal deterministic automaton that recognizes the language.

In [2], we introduced and studied a new complexity measure, namely the *sensing cost* of the language. Intuitively, the sensing cost of a language measures the detail with which a random input word needs to be read in order to decide membership in the language. Sensing has been studied in several other CS contexts. In theoretical CS, in methodologies such as PCP and property testing, we are allowed to sample or query only part of the input [8]. In more practical applications, mathematical tools in signal processing are used to reconstruct information based on compressed sensing [3], and in the context of data streaming, one cannot store in memory the entire input, and therefore has to approximate its properties according to partial "sketches" [12].

Our study in [2] considered regular and ω-regular languages, where sensing is defined as follows. Consider a deterministic automaton $\mathcal{A}$ over an alphabet 2^P, for a finite set P of *signals*. For a state q of $\mathcal{A}$, we say that a signal $p \in P$ is *sensed* in q if at least one transition taken from q depends on the truth value of p. The *sensing cost* of q is the number of signals it senses, and the sensing cost of a run is the average sensing cost of states visited along the run. We extend the definition to automata by assuming a given distribution of the inputs. The sensing cost of a language with respect to this distribution is then the infimum sensing cost of an automaton for the language. For simplicity, we focus on the uniform distribution, and we refer to the sensing cost of an automaton without parameterizing it by a distribution. As detailed in Remark 2.1, all our results can be extended to a setting with a parameterized distribution.

In [2], we showed that computing the sensing cost of a language can be done in polynomial time. We further showed that while in finite words the minimal sensing cost is always attained, this is not the case for infinite words. For example, recognizing the language L over $2^{\{p\}}$ of all words with infinitely many p's, one can give up sensing of p for unboundedly-long intervals, thus the sensing cost of L is 0, yet every deterministic automaton $\mathcal{A}$ that recognizes L must sense p infinitely often, causing the sensing cost of $\mathcal{A}$ to be strictly greater than 0.

In the context of formal methods, sensing has two appealing applications. The first is *monitoring*: we are given a computation and we have to decide whether it satisfies a specification. When the computations are over 2^P, we want to design a monitor that minimizes the expected average number of sensors used in the monitoring process. Monitoring is especially useful when reasoning about *safety* specifications [7]. There, every computation that violates the specification has a bad prefix – one all whose extensions are not in L. Hence, as long as the computation is a prefix of some word in L, the monitor continues to sense and examine the computation. Once a bad prefix is detected, the monitor declares an error and no further sensing is required. The second application is *synthesis*. Here, the set P of signals is partitioned into sets I and O of input and output signals, respectively. We are given a specification L over the alphabet $2^{I \cup O}$, and our goal is to construct an I/O *transducer* that realizes L. That is, for every sequence of assignments to the input signals, the transducer generates a sequence of assignments to the output signals so that the obtained computation is in L [13]. Our goal is to construct a transducer that minimizes the expected average number of sensors (of input signals) that are used along the interaction.

The definition of sensing cost in [2] falls short in the above two applications. For the first, the definition in [2] does not distinguish between words in the language and words not in the language, whereas in monitoring we care only for words in the language. In particular, according to the definition in [2], the sensing cost of a safety language is always 0. For the second, the definition in [2] considers automata and does not partition P into I and O, whereas synthesis refers to I/O-transducers. Moreover, unlike automata, correct transducers generate only computations in the language, and they need not generate all words in the language – only these that ensure receptiveness with respect to all sequences of inputs.

In this work we study sensing in the context of monitoring and synthesis. We suggest definitions that capture the intuition of "required number of sensors" in these settings and solve the problems of generating monitors and transducers that minimize sensing. For both settings, we focus on safety languages.

Consider, for example, a traffic monitor that has access to various sensors on roads and whose goal is to detect accidents. Once a road accident is detected, an alarm is raised to the proper authorities and the monitoring is stopped until the accident has been taken care of. The monitor can read the speed of cars along the roads, as well as the state of traffic lights. An accident is detected when some cars do not move even-though no traffic light is stopping them. Sensing the speed of every car and checking every traffic light requires huge sensing. Our goal is to find a monitor that minimizes the required sensing and still detects all accidents. In the synthesis setting, our goal is extended to designing a transducer that controls the traffic lights according to the speed of the traffic in each direction, and satisfies some specification (say, give priority to slow traffic), while minimizing the sensing of cars.

We can now describe our model and results. Let us start with monitoring. Recall that the definition of sensing in [2] assumes a uniform probability on the assignments to the signals, whereas in monitoring we want to consider instead more intricate probability spaces – ones that restrict attention to words in the language. As we show, there is more than one way to define such probability spaces, each leading to a different measure. We study two such measures. In the first, we sample a word randomly, letter by letter, according to a given distribution, allowing only letters that do not generate bad prefixes. In the second, we construct a sample space directly on the words in the language. We show that in both definitions, we can compute the sensing cost of the language in polynomial time, and that the minimal sensing cost is attained by a minimal-size automaton. Thus, luckily enough, even though different ways in which a computation may be given in an online manner calls for two definitions of sensing cost, the design of a minimally-sensing monitor is the same in the two definitions.

Next, we proceed to study sensing for synthesis. The main challenge there is that we no longer need to consider all words in the language. Also, giving up sensing has a flavor of synthesis with incomplete information [9]: the transducer has to realize the specification no matter what the incomplete information is. This introduces a new degree of freedom, which requires different techniques than those used in [2]. In particular, while a minimal-size transducer for a safety language can be defined on top of the state space of a minimal-size deterministic automaton for the language, this is not the case when we seek minimally-sensing transducers. This is different also from the results in [2] and even these in the monitoring setting, where a minimally-sensing automaton or monitor for a safety language coincides with the minimal-size automaton for it. In fact, we show that a minimally-sensing transducer for a safety language might be exponentially bigger than a minimal-size automaton for the language. Consequently, the problems of computing the minimal sensing cost and finding a minimally-sensing transducer are EXPTIME-complete even for specifications given by means of deterministic automata. On the positive side, a transducer that attains the minimal sensing cost always exists.

2 Preliminaries

Automata and Transducers

A *deterministic automaton on infinite words* is $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, \alpha \rangle$, where Q is a finite set of states, $q_0 \in Q$ is an initial state, $\delta : Q \times \Sigma \nrightarrow Q$ is a partial transition function, and α is an acceptance condition. We sometimes refer to δ as a relation $\Delta \subseteq Q \times \Sigma \times Q$, with $\langle q, \sigma, q' \rangle \in \Delta$ iff $\delta(q, \sigma) = q'$. A run of $\mathcal{A}$ on a word $w = \sigma_1 \cdot \sigma_2 \cdots \in \Sigma^\omega$ is a sequence of states $q_0, q_1, \ldots$ such that $q_{i+1} = \delta(q_i, \sigma_{i+1})$ for all $i \geqslant 0$. Note that since δ is deterministic and partial, $\mathcal{A}$ has at most one run on a word. A run is accepting if it satisfies the acceptance condition. A word $w \in \Sigma^\omega$ is accepted by $\mathcal{A}$ if $\mathcal{A}$ has an accepting run on w. The language of $\mathcal{A}$, denoted $L(\mathcal{A})$, is the set of words that $\mathcal{A}$ accepts. We denote by

$\mathcal{A}^q$ the automaton $\mathcal{A}$ with the initial state set to q.

In a deterministic *looping* automaton (DLW), every run is accepting. Thus, a word is accepted if there is a run of the automaton on it.[1] Since every run is accepting, we omit the acceptance condition and write $\mathcal{A} = \langle \Sigma, Q, q_0, \delta \rangle$.

For finite sets I and O of input and output signals, respectively, an *I/O transducer* is $\mathcal{T} = \langle I, O, Q, q_0, \delta, \rho \rangle$, where Q is a finite set of states, $q_0 \in Q$ is an initial state, $\delta : Q \times 2^I \rightarrow Q$ is a total transition function, and $\rho : Q \rightarrow 2^O$ is a labeling function on the states. The run of $\mathcal{T}$ on a word $w = i_0 \cdot i_1 \cdots \in (2^I)^\omega$ is the sequence of states $q_0, q_1, \ldots$ such that $q_{k+1} = \delta(q_k, i_k)$ for all $k \geqslant 0$. The *output* of $\mathcal{T}$ on w is then $o_1, o_2, \ldots \in (2^O)^\omega$ where $o_k = \rho(q_k)$ for all $k \geqslant 1$. Note that the first output assignment is that of q_1, and we do not consider $\rho(q_0)$. This reflects the fact that the environment initiates the interaction. The *computation of $\mathcal{T}$ on w* is then $\mathcal{T}(w) = i_0 \cup o_1, i_1 \cup o_2, \ldots \in (2^{I \cup O})^\omega$.

Note that the structure of each I/O-transducer $\mathcal{T}$ induces a DLW $\mathcal{A}_\mathcal{T}$ over the alphabet 2^I with a total transition relation. Thus, the language of the DLW is $(2^I)^\omega$, reflecting the receptiveness of $\mathcal{T}$.

Safety Languages

Consider a language $L \subseteq \Sigma^\omega$. A finite word $x \in \Sigma^*$ is a *bad prefix* for L if for every $y \in \Sigma^\omega$, we have that $x \cdot y \notin L$. That is, x is a bad prefix if all its extensions are words not in L. The language L is then a *safety* language if every word not in L has a bad prefix. For a language L, let $\mathit{pref}(L) = \{x \in \Sigma^* : \text{ there exists } y \in \Sigma^\omega \text{ such that } x \cdot y \in L\}$ be the set of prefixes of words in L. Note that each word in Σ^* is either in $\mathit{pref}(L)$ or is a bad prefix for L. Since the set $\mathit{pref}(L)$ for a safety language L is *fusion closed* (that is, a word is in L iff all its prefixes are in $\mathit{pref}(L)$), an ω-regular language is safety iff it can be recognized by a DLW [15].

Consider a safety language L over sets I and O of input and output signals. We say that L is *I/O-realizable* if there exists an I/O transducer $\mathcal{T}$ all whose computations are in L. Thus, for every $w \in (2^I)^\omega$, we have that $\mathcal{T}(w) \in L$. We then say that $\mathcal{T}$ *I/O-realizes* L. When I and O are clear from the context, we omit them. The *synthesis* problem gets as input a safety language L over I and O, say by means of a DLW, and returns an I/O-transducer that realizes L or declares that L is not I/O-realizable.

Sensing

In [2], we defined regular sensing as a measure for the number of sensors that need to be operated in order to recognize a regular language. We study languages over an alphabet $\Sigma = 2^P$, for a finite set P of signals. A letter $\sigma \in \Sigma$ corresponds to a truth assignment

[1] For readers familiar with the Büchi acceptance condition, a looping automaton is a special case of Büchi with $\alpha = Q$.

to the signals, and sensing a signal amounts to knowing its assignment. Describing sets of letters in Σ, it is convenient to use Boolean assertions over P. For example, when $P = \{a, b\}$, the assertion $\neg b$ stands for the set $\{\emptyset, \{a\}\}$ of two letters.

For completeness, we bring here the definitions from [2]. Consider a language L and a deterministic automaton $\mathcal{A} = \langle 2^P, Q, q_0, \delta, \alpha\rangle$ such that $L(\mathcal{A}) = L$. We assume that δ is total. For a state $q \in Q$ and a signal $p \in P$, we say that p is *sensed in* q if there exists a set $S \subseteq P$ such that $\delta(q, S\backslash\{p\}) \neq \delta(q, S \cup \{p\})$. Intuitively, a signal is sensed in q if knowing its value may affect the destination of at least one transition from q. We use $sensed(q)$ to denote the set of signals sensed in q. The *sensing cost* of a state $q \in Q$ is $scost(q) = |sensed(q)|$. [2]

For a finite run $r = q_1, \ldots, q_m$ of $\mathcal{A}$, we define the sensing cost of r, denoted $scost(r)$, as $\frac{1}{m}\sum_{i=0}^{m-1} scost(q_i)$. That is, $scost(r)$ is the average number of sensors that $\mathcal{A}$ uses during r. Now, for a finite word w, we define the sensing cost of w in $\mathcal{A}$, denoted $scost_{\mathcal{A}}(w)$, as the sensing cost of the run of $\mathcal{A}$ on w. Finally, the sensing cost of $\mathcal{A}$ is the expected sensing cost of words of length that tends to infinity, where we assume that the letters in Σ are uniformly distributed (see Remark 2.1 below). Thus, $scost(\mathcal{A}) = \lim_{m\to\infty} |\Sigma|^{-m} \sum_{w\in\Sigma^m} scost_{\mathcal{A}}(w)$.

Note that the definition applies to automata on both finite and infinite words, and it corresponds to the closed setting: the automaton gets as input words over 2^P and uses sensors in order to monitor the input words and decide their membership in L. We define the *sensing cost* of a language L to be the minimal cost of an automaton for L. A-priori, the minimal cost might not be attained by a single automaton, thus we define $scost(L) = \inf\{scost(\mathcal{A}) : \mathcal{A} \text{ is an automaton for } L\}$.

Remark 2.1 (On the choice of uniform distribution) *The choice of a uniform distribution on the letters in Σ may be unrealistic in practice. Indeed, in real scenarios, the distribution on the truth assignments to the underlying signals may be complicated. Generally, such a distribution can be given by a Markov chain (in monitoring) or by an MDP (in synthesis). As it turns out, adjusting our setting and algorithms to handle such distributions involves only a small technical elaboration, orthogonal to the technical challenges that exists already in a uniform distribution.*

Accordingly, throughout the paper we assume a uniform distribution on the truth assignments to the signals. In Appendix B we describe how our setting and algorithms are extended to the general case. □

The definition of sensing in [2] essentially considers the sensing required in the Ergodic SCC of a deterministic automaton for the language. Since in safety languages, the

[2] We note that, alternatively, one could define the *sensing level* of states, with $slevel(q) = \frac{scost(q)}{|P|}$. Then, for all states q, we have that $slevel(q) \in [0, 1]$. All our results hold also for this definition, simply by dividing the sensing cost by $|P|$.

Ergodic SCCs are accepting or rejecting sinks, which require no sensing, we have the following, which implies that the definition in [2] is not too informative for safety languages.

Lemma 2.2 *For every safety language $L \subseteq \Sigma^\omega$, we have $scost(L) = 0$.*

Markov Chains and Decision Processes

A Markov chain $\mathcal{M} = \langle S, P \rangle$ consists of a finite state space S and a stochastic transition matrix $P : S \times S \to [0, 1]$. That is, for all $s \in S$, we have $\sum_{s' \in S} P(s, s') = 1$. Given an initial state s_0, consider the vector v^0 in which $v^0(s_0) = 1$ and $v^0(s) = 0$ for every $s \neq s_0$. The *limiting distribution* of $\mathcal{M}$ is $\lim_{n\to\infty} \frac{1}{n} \sum_{m=0}^{n} v^0 P^m$. The limiting distribution satisfies $\pi P = \pi$, and can be computed in polynomial time [5].

A Markov decision process (MDP) is $\mathcal{M} = \langle S, s_0, (A_s)_{s\in S}, \mathrm{P}, \mathit{cost} \rangle$ where S is a finite set of states, $s_0 \in S$ is an initial state, A_s is a finite set of actions that are available in state $s \in S$. Let $A = \bigcup_{s\in S} A_s$. Then, $\mathrm{P} : S \times A \times S \nrightarrow [0, 1]$ is a partial transition probability function, defining for every two states $s, s' \in S$ and action $a \in A_s$, the probability of moving from s to s' when action a is taken. Accordingly, $\sum_{s' \in S} \mathrm{P}(s, a, s') = 1$. Finally, $\mathit{cost} : S \times A \nrightarrow \mathbb{N}$ is a partial cost function, assigning each state s and action $a \in A_s$, the cost of taking action a in state s.

An MDP can be thought of as a game between a player who chooses the actions and nature, which acts stochastically according to the transition probabilities.

A *policy* for an MDP $\mathcal{M}$ is a function $f : S^* \times S \to A$ that outputs an action given the history of the states, such that for $s_0, \ldots, s_n$ we have $f(s_0, \ldots, s_n) \in A_{s_n}$. Policies correspond to the strategies of the player. The *cost* of a policy f is the expected average cost of a random walk in $\mathcal{M}$ in which the player proceeds according to f. Formally, for $m \in \mathbb{N}$ and for a sequence of states $\tau = s_0, \ldots, s_{m-1}$, we define $\mathrm{P}_f(\tau) = \prod_{i=1}^{m-1} \mathrm{P}(s_{i-1}, f(s_0 \cdots s_{i-1}), s_i)$. Then, $\mathit{cost}_m(f, \tau) = \frac{1}{m} \sum_{i=1}^{m} \mathit{cost}(s_i, f(s_1 \cdots s_i))$ and we define the cost of f as $\mathit{cost}(f) = \liminf_{m\to\infty} \frac{1}{m} \sum_{\tau : |\tau| = m} \mathit{cost}_m(f, \tau) \cdot \mathrm{P}_f(\tau)$.

A policy is *memoryless* if it depends only on the current state. We can describe a memoryless policy by $f : S \to A$. A memoryless policy f induces a Markov chain $\mathcal{M}^f = \langle S, P_f \rangle$ with $P_f(s, s') = \mathrm{P}(s, f(s), s')$. Let π be the limiting distribution of $\mathcal{M}^f$. It is not hard to prove that $\mathit{cost}(f) = \sum_{s\in S} \pi_s \mathit{cost}(s, f(s))$. Let $\mathit{cost}(\mathcal{M}) = \inf\{\mathit{cost}(f) : f$ is a policy for $\mathcal{M}\}$. That is, $\mathit{cost}(M)$ is the expected cost of a game played on $\mathcal{M}$ in which the player uses an optimal policy.

Theorem 2.3 *Consider an MDP $\mathcal{M}$. Then, $\mathit{cost}(\mathcal{M})$ can be attained by a memoryless policy, which can be computed in polynomial time.*

3 Monitoring

As described in Section 2, the definition of sensing in [2] takes into an account all words in $(2^P)^\omega$, regardless their membership in the language. In monitoring, we restrict attention to words in the language, as once a violation is detected, no further sensing is required. In particular, in safety languages, violation amounts to a detection of a bad prefix, and indeed safety languages are the prominent class of languages for which monitoring is used [7].

As it turns out, however, there are many approaches to define the corresponding probability space. We suggest here two. Let $\mathcal{A}$ be a DLW and let $L = L(\mathcal{A})$.

1. **[Letter-based]** At each step, we uniformly draw a "safe" letter – one with which we are still generating a word in $\mathit{pref}(L)$, thereby iteratively generating a random word in L.

2. **[Word-based]** At the beginning, we uniformly draw a word in L.

We denote the sensing cost of $\mathcal{A}$ in the letter- and word-based approaches $\mathit{lcost}(\mathcal{A})$ and $\mathit{wcost}(\mathcal{A})$, respectively. The two definitions yield two different probability measures on L, as demonstrated in Example 3.1 below.

Example 3.1 *Let $P = \{a\}$ and consider the safety language $L = a^\omega + (\neg a) \cdot (\mathit{True})^\omega$. That is, if the first letter is $\{a\}$, then the suffix should be $\{a\}^\omega$, and if the first letter is $\emptyset$, then all suffixes result in a word in L. Consider the* DLW *$\mathcal{A}$ for L in Figure 1.*

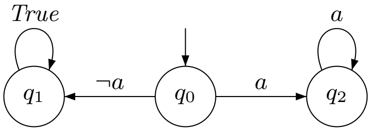


Figure 1: A DLW for $a^\omega + (\neg a) \cdot (\mathit{True})^\omega$.

In the letter-based definition, we initially draw a letter from $2^{\{a\}}$ uniformly, i.e., either a or $\neg a$ w.p. $\frac{1}{2}$. If we draw $\neg a$, then we move to q_1 and stay there forever. If we draw a, then we move to q_2 and stay there forever. Since $\mathit{scost}(q_1) = 0$ and $\mathit{scost}(q_2) = 1$, and we reach q_1 and q_2 w.p $\frac{1}{2}$, we get $\mathit{lcost}(\mathcal{A}) = \frac{1}{2}$.

In the word-based definition, we assign a uniform probability to the words in L. In this case, almost all words are not a^ω, and thus the probability of a^ω is 0. This means that we will get to q_1 w.p. 1, and thus $\mathit{wcost}(\mathcal{A}) = 0$. □

As a more realistic example, recall our traffic monitor in Section 1. There, the behavior of the cars is the random input, and the two approaches can be understood as follows. In the letter-based approach, we assume that the drivers do their best to avoid accidents regardless of the history of the traffic and the traffic lights so far. Thus, after every safe prefix, we assume that the next input is also safe. In the word-based approach, we assume that the city is planned well enough to avoid accidents. Thus, we a-priori set the distribution to safe traffic behaviors according to their likelihood.

We now define the two approaches formally.

The Letter-Based Approach Consider a DLW $\mathcal{A} = \langle \Sigma, Q, \delta, q_0 \rangle$. For a state $q \in Q$, let $avail(q)$ be the set of letters available in q, namely letters that do not cause $\mathcal{A}$ to get stuck. Formally, $avail(q) = \{\sigma \in \Sigma : \delta(q, \sigma)$ is defined $\}$. We model the drawing of available letters by the Markov chain $\mathcal{M}_{\mathcal{A}} = \langle Q, P \rangle$, where the probability of a transition from state q to state q' in $\mathcal{M}_{\mathcal{A}}$ is $P(q, q') = \frac{|\{\sigma \in \Sigma : \delta(q,\sigma) = q'\}|}{|avail(q)|}$. Let π be the limiting distribution of $\mathcal{M}_{\mathcal{A}}$. We define $lcost(\mathcal{A}) = \sum_{q \in Q} \pi(q) \cdot scost(q)$.

Since computing the limiting distribution can be done in polynomial time, we have the following.

Theorem 3.1 *Given a* DLW $\mathcal{A}$*, the sensing cost* $lcost(\mathcal{A})$ *can be calculated in polynomial time.*

The Word-Based Approach Consider a DLW $\mathcal{A} = \langle 2^P, Q, q_0, \delta \rangle$ recognizing a non-empty safety language L. From [2], we have $scost(\mathcal{A}) = \lim_{n \to \infty} \frac{1}{|\Sigma|^n} \sum_{u \in \Sigma^n} scost_{\mathcal{A}}(u)$, which is proven to coincide with $\mathbb{E}[scost_{\mathcal{A}}(u)]$ where $\mathbb{E}$ is the expectation with respect to the standard measure on Σ^ω. Our goal here is to replace this standard measure with one that restricts attention to words in L. Thus, we define $wcost(\mathcal{A}) = \mathbb{E}[scost(u) \mid u \in L]$. For $n \geqslant 0$, let $pref(L, n)$ be the set of prefixes of L of length n. Formally, $pref(L, n) = pref(L) \cap \Sigma^n$. As in the case of the standard measure, the expectation-based definition coincides with one that that is based on a limiting process: $wcost(\mathcal{A}) = \lim_{n \to \infty} \frac{1}{|pref(L,n)|} \sum_{u \in pref(L,n)} scost_{\mathcal{A}}($
Thus, the expressions for $scost$ and $wcost$ are similar, except that in the expectation-based definition we add conditional probability, restricting attention to words in L, and in the limiting process we replace Σ^n by $pref(L, n)$.

Note that the term $\frac{1}{|pref(L,n)|}$ is always defined, as L is a non-empty safety language. In particular, the expectation is well defined even if L has measure 0 in Σ^ω.

Theorem 3.2 *Given a DLW $\mathcal{A}$, we can compute $wcost(\mathcal{A})$ in polynomial time.*

Proof: We will use here formal power series on one variable z, a classical tool for graph and automata combinatorics. They can be thought of as polynomials of infinite degree.

For states $p, q \in Q$ and for $n \in \mathbb{N}$, let $\#paths(p, q, n)$ denote the number of paths (each one labeled by a distinct word) of length n from p to q in $\mathcal{A}$. We define the generating functions: $C_{p,q}(z) = \sum_{n \in \mathbb{N}} \#paths(p, q, n) z^n$ and $F_q(z) = C_{q_0,q}(z) \sum_{p \in Q} C_{q,p}(z)$. Let $[z^n]F_q(z)$ be the coefficient of z^n in $F_q(z)$. By the definition of $C_{q_0,q}$, we get

$$[z^n]F_q(z) = \sum_{k=0}^{n} \#paths(q_0, q, k) \sum_{p \in Q} \#paths(q, p, n-k).$$

Therefore, $[z^n]F_q(z)$ is the total number of times the state q is used when listing all paths of length n from q_0.

Thus, we have $\sum_{u\in pref(L,n)} scost(u) = \frac{1}{n}\sum_{q\in Q} scost(q)[z^n]F_q(z)$. Finally, let $S(z) = \sum_{p\in p} C_{q_0,p}(z)$. Then, $wcost(\mathcal{A}) = \lim_{n\to\infty} \frac{1}{n\cdot[z^n]S(z)} \sum_{q\in Q} scost(q)[z^n]F_q(z)$. In Appendix A.1 we use techniques from [4] and [14] to compute the latter limit in polynomial time, by asymptotic estimations of the coefficients, thus concluding the proof. □

Sensing cost of languages

For a safety language L, we define $lcost(L) = \inf\{lcost(\mathcal{A}) : \mathcal{A}$ is a DLW for $L\}$, and similarly for $wcost(L)$. Different DLWs for a language L may have different sensing costs. We show that the minimal sensing cost in both approaches is attained at the minimal-size DLW. We first need some definitions and notations.

Consider a safety language $L \subseteq \Sigma^\omega$. For two finite words u_1 and u_2, we say that u_1 and u_2 are *right L-indistinguishable*, denoted $u_1 \sim_L u_2$, if for every $z \in \Sigma^\omega$, we have that $u_1 \cdot z \in L$ iff $u_2 \cdot z \in L$. Thus, $\sim_L$ is the Myhill-Nerode right congruence used for minimizing DFAs. For $u \in \Sigma^*$, let $[u]$ denote the equivalence class of u in $\sim_L$ and let $\langle L\rangle$ denote the set of all equivalence classes. Each class $[u] \in \langle L\rangle$ is associated with the *residual language* $u^{-1}L = \{w : uw \in L\}$. Note that for safety languages, there is at most one class $[u]$, namely the class of bad prefixes, such that $u^{-1}L = \emptyset$. We denote this class $[\bot]$. When $L \neq \emptyset$ is a regular safety language, the set $\langle L\rangle$ is finite, and induces the *residual automaton* of L, defined by $\mathcal{R}_L = \langle \Sigma, \langle L\rangle\backslash\{[\bot]\}, \delta_L, [\epsilon]\rangle$, with $\delta_L([u], a) = [u \cdot a]$ for all $[u] \in \langle L\rangle\backslash\{[\bot]\}$ and $a \in \Sigma$ such that $[u \cdot a] \neq [\bot]$. The automaton $\mathcal{R}_L$ is well defined and is the unique minimal-size DLW for L.

Consider a DLW $\mathcal{A} = \langle \Sigma, Q, q_0, \delta\rangle$ such that $L(\mathcal{A}) = L$. For a state $s = [u] \in \langle L\rangle\backslash\{[\bot]\}$, we associate with s a set $states(\mathcal{A}, s) = \{q \in Q : L(\mathcal{A}^q) = u^{-1}L\}$. That is, $states(\mathcal{A}, s) \subseteq Q$ contains exactly all state that $\mathcal{A}$ can be in after reading a word that leads $\mathcal{R}_L$ to $[u]$.

The following claims are simple exercises.

Proposition 3.3 *Consider a safety language L and a DLW $\mathcal{A}$ for it.*

1. *The set $\{states(\mathcal{A}, s) : s \in \langle L\rangle\backslash\{[\bot]\}\}$ forms a partition of the states of $\mathcal{A}$.*

2. *For every state $s \in \langle L\rangle\backslash\{[\bot]\}$ of $\mathcal{R}_L$, letter $\sigma \in \Sigma$, and state $q \in states(\mathcal{A}, s)$, we have $\delta(q, \sigma) \in states(\mathcal{A}, \delta_L(s, \sigma))$.*

Lemma 3.4 *Consider a safety language $L \subseteq \Sigma^\omega$. For every* DLW *$\mathcal{A}$ with $L(\mathcal{A}) = L$, we have that $lcost(\mathcal{A}) \geqslant lcost(\mathcal{R}_L)$ and $wcost(\mathcal{A}) \geqslant wcost(\mathcal{R}_L)$*

Proof: We outline the key points in the proof for $lcost$. The arguments for $wcost$ are similar. For a detailed proof see Appendix A.2.

Recall that the states of $\mathcal{R}_L$ are $\langle L\rangle\backslash\{[\bot]\}$. We start by showing that for every $s \in \langle L\rangle\backslash\{[\bot]\}$ and for every $q \in states(\mathcal{A}, s)$ we have that $scost(q) \geqslant scost(s)$. Next, we consider the Markov chains $\mathcal{M}_{\mathcal{A}}$ and $\mathcal{M}_{\mathcal{R}_L}$. Using Proposition 3.3 we show that if π and τ are the limiting distributions of $\mathcal{M}_{\mathcal{A}}$ and $\mathcal{M}_{\mathcal{R}_L}$ respectively, then for every $s \in \langle L\rangle\backslash\{[\bot]\}$ we have that $\tau(s) = \sum_{q\in states(\mathcal{A},s)} \pi(q)$. Finally, since Q is partitioned by $\{states(\mathcal{A}, s)\}_s$ we conclude that $lcost(\mathcal{A}) \geqslant lcost(\mathcal{R}_L)$. □ Lemma 3.4 and Theorems 3.1 and 3.2 allow us to conclude with the following.

Theorem 3.5 *Given a* DLW $\mathcal{A}$*, we can compute* $lcost(L(\mathcal{A}))$ *and* $wcost(L(\mathcal{A}))$ *in polynomial time.*

Example 3.2 *Consider the DLW $\mathcal{A}$ over the alphabet $2^{\{a,b\}}$ appearing in Figure 2.*

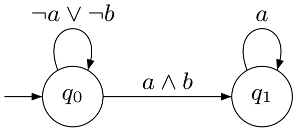


Figure 2: A DLW for $(\neg a \vee \neg b)^\omega + (\neg a \vee \neg b)^* \cdot (a \wedge b) \cdot a^\omega$.

Clearly, $\mathcal{A}$ is a minimal automaton for $L = (\neg a \vee \neg b)^\omega + (\neg a \vee \neg b)^* \cdot (a \wedge b) \cdot a^\omega$. By Lemma 3.4, we can calculate the sensing cost of $\mathcal{A}$ in order to find the sensing cost of L.

Clearly, $scost(q_0) = 2$ and $scost(q_1) = 1$. We start by computing $lcost(\mathcal{A})$. The corresponding Markov chain $M_{\mathcal{A}}$ has only one ergodic component $\{q_1\}$, so we obtain $lcost(\mathcal{A}) = scost(q_1) = 1$.

The computation of $wcost(\mathcal{A})$ is more intricate. In Appendix A.3 we show that $wcost(\mathcal{A}) = 2$. We remark that unlike in the other versions of sensing cost, transient components can play a role in $wcost$. In particular, If the self-loop on q_0 has been labeled by two rather than three letters, then we would have gotten $wcost(\mathcal{A}) = \frac{3}{2}$. □

4 Synthesis

In the setting of synthesis, the signals in P are partitioned into sets I and O of input and output signals. An I/O-transducer $\mathcal{T}$ senses only input signals and we define its sensing cost as the sensing cost of the DLW $\mathcal{A}_{\mathcal{T}}$ it induces.

We define the *I/O-sensing cost* of a realizable specification $L \in (2^{I\cup O})^\omega$ as the minimal cost of an I/O-transducer that realizes L. Thus, $scost_{I/O}(\mathcal{A}) = \inf\{scost(\mathcal{T}) : \mathcal{T}$ is an I/O-transducer that realizes $L\}$. In this section we consider the problem of finding a minimally-sensing I/O-transducer that realizes L.

The realizability problem for a DLW specifications can be solved in polynomial time. Indeed, given a DLW $\mathcal{A}$, we can view $\mathcal{A}$ as a game between a system, which controls the outputs, and an environment, which controls the inputs. We look for a strategy for the system that never reaches an undefined transition. This amounts to solving a turn-based safety game, which can be done in polynomial time.

When sensing is introduced, it is not enough for the system to win this game, as it now has to win while minimizing the sensing cost. Intuitively, not sensing some inputs introduces incomplete information to the game: once the system gives up sensing, it may not know the state in which the game is and knows instead only a set of states in which the game may be. In particular, unlike usual realizability, a strategy that minimizes the sensing need not use the state space of the DLW. We start with an example illustrating this.

Example 4.1 *Consider the* DLW $\mathcal{A}$ *appearing in Figure 3. The* DLW *is over* $I = \{p, q\}$ *and* $O = \{a\}$*. A realizing transducer over the structure of* $\mathcal{A}$ *(see* $\mathcal{T}_1$ *in Figure 4) senses* p *and* q*, responds with* a *if* $p \wedge q$ *was sensed and responds with* $\neg a$ *if* $\neg p \wedge \neg q$ *was sensed. In case other inputs are sensed, the response is arbitrary (denoted* $*$ *in the figure). As* $\mathcal{T}_1$ *demonstrates, every transducer that is based on the structure of* $\mathcal{A}$ *senses two input signals (both* p *and* q*) every second step, thus its sensing cost is* 1*. As demonstrated by the transducer* $\mathcal{T}_2$ *in Figure 5, it is possible to realize* $\mathcal{A}$ *with sensing cost of* $\frac{1}{2}$ *by only sensing* p *every second step. Indeed, knowing the value of* p *is enough in order to determine the output. Note that* $\mathcal{T}_2$ *may output sometimes* a *and sometimes* $\neg a$ *after reading assignments that causes* $\mathcal{A}$ *to reach* q_3*. Such a behavior cannot be exhibited by a transducer with the state-structure of* $\mathcal{A}$*.* □

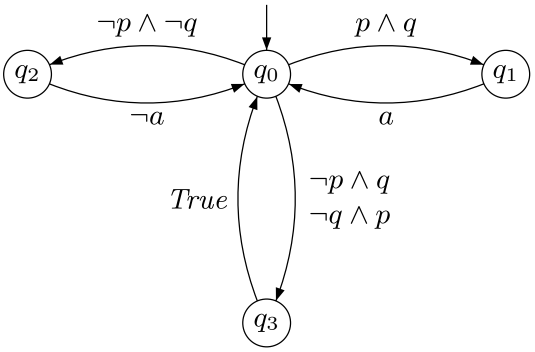


Figure 3: The DLW $\mathcal{A}$ in Example 4.1.

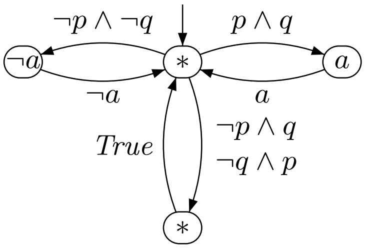


Figure 4: The transducer $\mathcal{T}_1$ for $\mathcal{A}$.

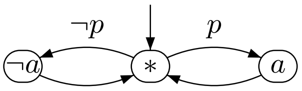


Figure 5: The transducer $\mathcal{T}_2$ for $\mathcal{A}$.

Solving games with incomplete information is typically done by some kind of a subset-construction, which involves an exponential blow up. Unlike usual games with incomplete information, here the strategy of the system should not only take care of the realizability but also decides which input signals should be sensed, where the goal is to obtain a minimally sensing transducer. In order to address these multiple objectives, we first construct an MDP in which the possible policies are all winning for the system, and corresponds to different choices of sensing. An optimal policy in this MDP then induces a minimally-sensing transducer.

Theorem 4.1 *Consider a* DLW $\mathcal{A}$ *over* $2^{I \cup O}$*. If* $\mathcal{A}$ *is realizable, then there exists an MDP* $\mathcal{M}$ *in which an optimal strategy corresponds to a minimally-sensing* I/O*-transducer that realizes* $\mathcal{A}$*. The MDP* $\mathcal{M}$ *has size exponential in* $|\mathcal{A}|$ *and can be computed in time exponential in* $|\mathcal{A}|$*.*

Proof: Consider a DLW $\mathcal{A} = \langle 2^{I \cup O}, Q, q_0, \delta \rangle$. We obtain from $\mathcal{A}$ an MDP $\mathcal{M} = \langle \mathcal{S}, \text{START}, A, \mathrm{P}, cost \rangle$, where $\mathcal{S} = (2^Q \times \{0,1,\perp\}^I) \cup \{\text{START}\}$, and $A = 2^I \times 2^O$. Intuitively, when $\mathcal{M}$ is in state $\langle S, \ell \rangle$, for $S \subseteq Q$ and $\ell : I \to \{0,1,\perp\}$, then $\mathcal{A}$ can be in every state in S, and for each input signal $b \in I$, we have that either b is true ($\ell(b) = 1$), b is false ($\ell(b) = 0$), or b is not sensed ($\ell(b) = \perp$). The action (o, i) means that we now output o and in the next state we will sense only inputs in i. For $\star \in \{\perp, 0, 1\}$, we define $\ell_\star = \{b \in I : \ell(b) = \star\}$.

We define the actions so that an action $\langle o, i \rangle$ is available in state $\langle S, \ell \rangle$ if for every $q \in S$ and $i' \subseteq \ell_\perp$, we have that $\delta(q, \ell_1 \cup i' \cup o)$ is defined. That is, an action is available if its o component does not cause $\mathcal{A}$ to get stuck no matter what the assignment to the signals that are not sensed is.

The transition probabilities are defined as follows. Consider a state $\langle S, \ell \rangle$, and an available action $\langle o, i \rangle$. Let $S' = \bigcup_{q \in S} \bigcup_{i' \subseteq \ell_\perp} \{\delta(q, \ell_1 \cup i' \cup o)\}$. Recall that by taking action $\langle o, i \rangle$, we decide that in the next state we will only sense signals in i. For $i \subseteq I$, we say that an assignment $\ell' : I \to \{0,1,\perp\}$ *senses* i if $\ell'_1 \cup \ell'_0 = i$. Note that there are $2^{|i|}$ assignments that sense i. Accordingly, we have $\mathrm{P}(\langle S, \ell \rangle, \langle o, i \rangle, \langle S', \ell' \rangle) = 2^{-|i|}$ for every $\ell' : I \to \{0,1,\perp\}$ that senses i. That is, a transition from $\langle S, \ell \rangle$ with $\langle o, i \rangle$ goes to the set of all possible successors of S under inputs that are consistent with ℓ and the output assignment o, and the ℓ' component is selected with uniform distribution among all assignments that sense i. The cost function depends on the number of signals we sense, thus $cost(\langle S, \ell \rangle) = |\ell_1 \cup \ell_0|$.

Finally, in the state START we only choose an initial set of input signals to sense. Thus, for every ℓ such that $\ell_1 \cup \ell_0$, we have $\mathrm{P}(\text{START}, \langle o, i \rangle, \langle \{q_0\}, \ell \rangle) = 2^{-|i|}$. Note that START is not reachable from any state in $\mathcal{M}$, and thus its cost is irrelevant. We arbitrarily set $cost(\text{START}) = 0$.

In Appendix A.4 we prove that $cost(\mathcal{M}) = scost_{I,O}(\mathcal{A})$ and that a minimal-cost policy f in $\mathcal{M}$ induces a minimally-sensing I/O-transducer that realizes $\mathcal{A}$. Intuitively, we prove this by showing a correspondence between transducers and policies, such that the sensing cost of a transducer $\mathcal{T}$ equals the value of the policy it corresponds to in $\mathcal{M}$.

Finally, we observe that the size of $\mathcal{M}$ is single exponential in the size of $\mathcal{A}$, and that we can construct $\mathcal{M}$ in time exponential in the size of $\mathcal{A}$. □

Theorem 4.2 *A minimally-sensing transducer for a realizable* DLW $\mathcal{A}$ *has size tightly exponential in* $|\mathcal{A}|$.

Proof: The upper bound follows from Theorem 2.3 applied to the MDP constructed in Theorem 4.1.

For the lower bound, we describe a family of realizable DLWs $\mathcal{A}_1, \mathcal{A}_2, \ldots$ such that for all $k \geqslant 1$, the DLW $\mathcal{A}_k$ has $1 + \sum_{i=1}^{k} p_i$ states, yet a minimally-sensing transducer for it requires at least $\prod_{i=1}^{k} p_i$ states, where $p_1, p_2, ...$ are prime numbers. Intuitively, $\mathcal{A}_k$ is constructed as follows. In the initial state q_{reset}, the inputs signals determine a number $1 \leqslant i \leqslant k$, and $\mathcal{A}_k$ moves to component i, which consists of a cycle of length p_i. In every state j in component i, the output signals must acknowledge that $\mathcal{A}_k$ is in state $0 \leqslant j < p_i$ of component i. Furthermore, we force a sensing of 1 in every state except for q_{reset} by requiring a signal to be acknowledged in every step. Finally, we can go back to q_{reset} only with a special output signal, which can be outputted only in state 0 of an i component.

Thus, a realizing transducer essentially only chooses which signals to read in q_{reset}. We show that 0 bits can be read, but in that case we need $\prod_{i=1}^{k} p_i$ states. Indeed, the transducer needs to keep track of the location in all the i components simultaneously, which means keeping track of the modulo from each p_i. Since every combination of such modulos is possible, the transducer needs $\prod_{i=1}^{k} p_i$ states. In Appendix A.5 we formalize this intuition. □

We now turn to study the complexity of the problem of finding a minimally-sensing transducer. By the construction in Theorem 4.1 and the polynomial time algorithm from Theorem 2.3, we have the following.

Theorem 4.3 *Consider a realizable* DLW $\mathcal{A}$ *over* $2^{I \cup O}$. *We can calculate* $cost_{I,O}(\mathcal{A})$ *and return a minimally-sensing* I/O*-transducer that realizes* $\mathcal{A}$ *in time exponential in* $|\mathcal{A}|$.

In order to complete the picture, we consider the corresponding decision problem. Given a DLW $\mathcal{A}$ over $2^{I \cup O}$ and a threshold γ, the sensing problem in the open setting is to decide whether $cost_{I,O}(\mathcal{A}) < \gamma$.

Theorem 4.4 *The sensing problem in the open setting is EXPTIME-complete.*

Proof: The upper bound follows from Theorem 4.3. For the lower bound, we show that the problem is EXPTIME hard even for a fixed γ. Given a DLW specification $\mathcal{A}$ over $2^{I \cup O}$, we show that it is EXPTIME-hard to decide whether there exists a transducer $\mathcal{T}$ that realizes $\mathcal{A}$ with $scost(\mathcal{T}) < 1$. We show a reduction from the problem of deciding the nonemptiness of an intersection of finite deterministic tree automata proved to be EXPTIME-hard in [6]. The idea is similar to that of Theorem 4.2, where a reset state is used to select an object, and a transducer can ignore the inputs in this state by using a response which is acceptable in every possible selected object.

A deterministic automaton on finite trees (DFT) is $\mathcal{U} = \langle \Sigma, Q, \delta, q_0, F\rangle$, where Σ is a finite alphabet, Q is a finite set of states, $q_0 \in Q$ is an initial state, $\delta : Q \times \Sigma \to Q \times Q$ is a transition function, and $F \subseteq Q$ is a set of accepting states. We refer to the left and right components of δ as $\delta_{\lhd}$ and $\delta_{\rhd}$. For example, when $\delta(q, \sigma) = \langle q_l, q_r\rangle$, we write $\delta_{\lhd}(q, \sigma) = q_l$. An DFT runs on Σ-*trees*. A (binary) Σ-tree is $T = \langle \tau, \ell\rangle$ where $\tau \subseteq \{\lhd, \rhd\}^*$ is prefix-closed: for every $x \cdot \sigma \in \tau$ it holds that $x \in \tau$, and $\ell : \tau \to \Sigma$ is a labeling function. For simplicity, we require that for every $x \in \tau$, either $\{x\lhd, x\rhd\} \subseteq \tau$, or $\{x\lhd, x\rhd\} \cap \tau =$, in which case x is a leaf. Given a tree $T = \langle \tau, \ell\rangle$, the *run* of $\mathcal{U}$ on T is a Q-tree $\langle \tau, \ell'\rangle$ where $\ell'(\epsilon) = q_0$, and for every $x \in \tau$ such that x is not a leaf, we have $\delta(\ell'(x), \ell(x)) = \langle \ell'(x\lhd), \ell'(x\rhd)\rangle$. A run is *accepting* if every leaf is labeled by an accepting state. A Σ-tree T is accepted by $\mathcal{U}$ if the run of $\mathcal{U}$ on T is accepting.

The nonempty-intersection problem gets as input DFTs $\mathcal{U}_1, \ldots, \mathcal{U}_n$, and decides whether their intersection is nonempty, that is $\bigcap_{t=1}^{n} L(\mathcal{U}_t) \neq \emptyset$. Given $\mathcal{U}_1, \ldots, \mathcal{U}_n$, we construct a specification DLW $\mathcal{A}$ such that $\bigcap_{t=1}^{n} L(\mathcal{U}_t) \neq \emptyset$ iff $scost(\mathcal{A}) < 1$. We assume w.l.o.g. that $L(\mathcal{U}_t) \neq \emptyset$ for all $1 \leqslant t \leqslant n$.

We construct $\mathcal{A}$ as follows. Initially, the inputs specify an index $1 \leqslant t \leqslant n$. Then, the transducer should respond with a tree in $L(\mathcal{U}_t)$. This is done by challenging the transducer with a branch in the tree, until some reset input signal is true, and the process repeats. Now, if $\bigcap_{t=1}^{n} L(\mathcal{U}_t) \neq \emptyset$, the transducer can ignore the input signals that specify the index t and just repeatedly output a tree in the intersection. On the other hand, if $\bigcap_{t=1}^{n} L(\mathcal{U}_t) = \emptyset$, the transducer must sense some information about the specified index.[3]

We now formalize this intuition. For $1 \leqslant t \leqslant n$, let $\mathcal{U}_t = \langle 2^J, Q^t, \delta^t, q_0^t, F^t\rangle$. Note that we assume w.l.o.g that the alphabet of all the DFTs is 2^J. We construct a specification DLW $\mathcal{A} = \langle 2^{I\cup O}, Q, q_0, \delta\rangle$ as follows. The set of states of $\mathcal{A}$ is $Q = \bigcup_{t=1}^{n} Q^t \cup \{\text{RESET}\}$. Assume w.l.o.g that $n = 2^k$ for some $k \in \mathbb{N}$. We define $I = \{b_1, \ldots, b_k\} \cup \{dI\}$ and $O = J \cup \{dO, e\}$. The input signal dI and the output signal dO denote the direction of branching in the tree. For clarity, in an input letter $i \in I$ we write $i(dI) = \lhd$ (and $i(dI) = \rhd$) to indicate that $dI \notin i$ (and $dI \in i$). We use a similar notation for dO.

We define the transition function as follows. In state RESET, we view the inputs $b_1, \ldots, b_k$ as a binary encoding of a number $t \in \{1, \ldots, n\}$. Then, $\delta(\text{RESET}, t) = q_0^t$. Next, consider a state $q \in Q^t$, and consider letters $i \subseteq I$ and $o \subseteq O$. We define δ as follows:

$$\delta(q, i \cup o) = \begin{cases} \text{RESET} & q \in F \wedge e \in o \wedge o(dO) = i(dI) \\ \delta^t_{\lhd}(q, o \cap J) & e \notin o \wedge o(dO) = i(dI) = \lhd \\ \delta^t_{\rhd}(q, o \cap J) & e \notin o \wedge o(dO) = i(dI) = \rhd \end{cases}$$

[3] Note that since a tree in the intersection of DFTs may be exponentially bigger than the DFTs, the lower bound here also suggests an alternative lower bound to the exponential size of a minimally-sensed transducer, now with a polynomial set of signals (as opposed to the proof of Theorem 4.2).

Note that $\delta(q, i \cup o)$ is undefined when $o(dO) \neq i(dI)$ or when $q \notin F$ and $e \in o$. Intuitively, in state RESET, an index $1 \leqslant t \leqslant n$ is chosen. From then on, in a state $q \in Q^t$, we simulate the run of $\mathcal{U}_t$ on the left or right branch of the tree, depending on the signal dI. The next letter is outputted in o, and additionally, we require that dO matches dI.

We claim that $scost(\mathcal{A}) < 1$ iff $\bigcap_{t=1}^{n} L(\mathcal{U}_t) \neq \emptyset$. In the first direction, assume that $\bigcap_{t=1}^{n} L(\mathcal{U}_t) \neq \emptyset$, and let T be a tree such that $T \in \bigcap_{t=1}^{n} L(\mathcal{U}_t) \neq \emptyset$. Consider the following transducer $\mathcal{T}$: in the state RESET it does not sense any inputs, and then it outputs a branch of T according to the signal dI, while always acknowledging the dI bit with the correct dO. When the end of the branch is reached, it outputs e. Since T is accepted by every DFT U^t, it follows that $\mathcal{T}$ realizes $\mathcal{A}$. Moreover, let l be the longest branch in T, then every l steps at most, $\mathcal{T}$ visits a state corresponding to RESET, in which it senses nothing. Thus, $\mathcal{T}$ senses 1 for at most l steps, and then 0. It follows that $scost(\mathcal{T}) \leqslant \frac{l}{l+1} = 1 - \frac{1}{l+1} < 1$.

Conversely, observe that in every state $q \in Q \backslash \{\text{RESET}\}$, a realizing transducer must sense at least 1 signal, namely dI. Thus, the only way to get sensing cost of less than 1 is to visit RESET infinitely often (in fact, with bounded sparsity), and to sense 0 in RESET. However, sensing 0 in RESET means that the next state could be the initial state of any of the n DFTs. Moreover, visiting RESET again means that at some point e was outputted in an accepting state of one of the DFTs. Thus, the transducer outputs a tree that is accepted in every DFT, so $\bigcap_{t=1}^{n} L(\mathcal{U}_t) \neq \emptyset$.

Finally, observe that the reduction is clearly polynomial, and thus we conclude that deciding whether $scost(\mathcal{A}) < 1$ is EXPTIME-hard. □

5 Discussion and Future Research

Sensing is a basic measure of the complexity of monitoring and synthesis. In monitoring safety properties, the definition of sensing presented in [2] is not informative, as it gives sensing cost 0 to properties that are satisfied with probability 0. We argue that in the context of monitoring, the definition of sensing cost should consider only computations that satisfy the property, and we study the complexity of computing the sensing cost of a property in the new definition. We distinguish between two approaches to define a probabilistic measure with respect to the set of computations that satisfy a property. We show that while computing the sensing cost according to the new definitions is technically more complicated than in [2], the minimal sensing is still attained by a minimal-size automaton, and it can still be computed in polynomial time.

In synthesis, we introduce a new degree of freedom, namely choosing the outputs when realizing a specification. We study the complexity of finding a minimal-sensing transducer for safety specifications. We show that the minimal-sensing transducer is not necessarily minimal in size. Moreover, interestingly, unlike the case of traditional synthesis, a minimal-sensing transducer need not even correspond to a strategy embodied in the specification deterministic automaton. On the positive side, we show that a minimal-sensing transducer always exists (for a realizable specification) and that its size is at most exponential in the size of the minimal-size transducer. We also provided matching lower bounds.

We now turn to discuss some future directions for research.

Non-safety properties We focus on safety properties. The study in [2] completes the monitoring picture for all other ω-regular properties. We plan to continue the study of synthesis of ω-regular properties. An immediate complication in this setting is that a finite minimal-sensing transducer does not always exists. Indeed, even in the monitoring setting studied in [2], a minimal-sensing automaton does not always exist.

A trade-off between sensing and quality Reducing the sensing cost of a transducer can often be achieved by *delaying* the sensing of some letter, thus sensing it less often. This, however, means that eventualities may take longer to be fulfilled, resulting in transducers of lower quality [1]. We plan to formalize and study the trade-off between the sensing and quality and relate it to the trade-offs between size and sensing, as well as between size and quality.

Acknowledgment We thank Elie de Panafieu for helpful discussions.

References

[1] S. Almagor, U. Boker, and O. Kupferman. Discounting in LTL. In *20th TACAS*, 2014.

[2] S. Almagor, D. Kuperberg, and O. Kupferman. Regular sensing. In *34th FSTTCS*, pages 161–173, 2014.

[3] D.L. Donoho. Compressed sensing. *IEEE Trans. Inform. Theory*, 52:1289–1306, 2006.

[4] P. Flajolet and R. Sedgewick. Analytic combinatorics: functional equations, rational and algebraic functions. 2001.

[5] C. Grinstead and J. Laurie Snell. 11:Markov chains. In *Introduction to Probability*. American Mathematical Society, 1997.

[6] J. Goubault. Rigid E-Unifiability is DEXPTIME-Complete. In *9th LICS*, pages 498–506, 1994.

[7] K. Havelund and G. Rosu. Efficient monitoring of safety properties. *Software Tools for Technology Transfer*, 6(2):18–173, 2004.

[8] G. Kindler. *Property Testing, PCP, and Juntas*. PhD thesis, Tel Aviv University, 2002.

[9] O. Kupferman and M.Y. Vardi. Synthesis with incomplete information. In *Advances in Temporal Logic*, pages 109–127. Kluwer Academic Publishers, 2000.

[10] E. Kushilevitz and N. Nisan. *Communication complexity*. Cambridge University Press, 1997.

[11] C. Mauduit and A. Sárköz. On finite pseudorandom binary sequences. i. measure of pseudorandomness, the legendre symbol. *Acta Arith.*, 82(4):365–377, 1997.

[12] S. Muthukrishnan. Theory of data stream computing: where to go. In *Proc. 30th PODS*, pages 317–319, 2011.

[13] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. 16th POPL*, pages 179–190, 1989.

[14] M.F. Roy and A. Szpirglas. Complexity of the computation on real algebraic numbers. *J. Symb. Comput.*, 10(1):39–52, 1990.

[15] A.P. Sistla. Safety, liveness and fairness in temporal logic. *Formal Aspects of Computing*, 6:495–511, 1994.

A Appendix

A.1 Proof of Theorem 3.2

By [4], for every $p, q \in Q$, we can compute in polynomial time (using standard algorithms on matrices) rational expressions for $C_{p,q}(z)$. The base case is for computing coefficients in the same irreducible aperiodic SCC represented by a matrix M: it suffices to compute the matrix $R(z) = (Id - zM)^{-1}$, its coefficient (p, q) is $C_{p,q}(z)$. For instance if $\{p\}$ is a SCC in $\mathcal{A}$ with a self-loop labeled by k letters, then $C_{p,p}(z) = \frac{1}{1-kz}$. Other $C_{p,q}$ are then computed from this base case via standard operations on rational functions. In particular, from [4] there is a period $d \leqslant |Q|$ such that for every $i \in \{0, \ldots, d-1\}$ and for all rational functions $Q(z) \in \{S(z)\} \cup \bigcup_{q \in Q} \{F_q(z)\}$ considered here, we can compute in polynomial time γ, k, and λ such that $[z^{nd+i}]Q(z) \sim \gamma(nd+i)^k \lambda^{nd+i}$ (where for functions $f, g : \mathbb{N} \to \mathbb{R}$ we have $f(n) \sim g(n)$ iff $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 1$). We remind the formula that allows us to do so. Let $Q(z) = \frac{A(z)+B(z)\left(1-\frac{z}{r}\right)^{-j}}{z^i}$ be a rational function of convergence radius r, where $i, j \in \mathbb{N}$, $A(z)$ and $B(z)$ have convergence radius strictly greater than r, and $B(r) \neq 0$. Then we have

$$[z^n]Q(z) \sim \frac{B(r)}{(j-1)! \cdot r^i} n^{j-1}(1/r)^n.$$

Notice that r will in general be a real algebraic number, that will be represented in our algorithm as a root of a polynomial with integer coefficients. Standard operations as sum, product, and comparisons on algebraic numbers (represented by polynomials) can be done in polynomial time, using techniques as described in [14].

Therefore, we can compute asymptotic equivalents of the form $\alpha \cdot n^k \cdot \lambda^n$ with α, λ real algebraic and $k \in \mathbb{N}$ [4], for both $[z^n]S(z)$ and $\frac{1}{n}\sum_{q \in Q} \mathit{scost}(q)[z^n]F_q(z)$, performing an averaging operation if $d > 1$. Finally, we can compute the wanted limit thanks to these asymptotic equivalents, thereby achieving the polynomial-time computation of $\mathit{wcost}(\mathcal{A})$.

A.2 Detailed proof of Lemma 3.4

We start with lcost. Consider a finite word $u \in \Sigma^*$ that is not a bad prefix for L. After reading u, the DLW $\mathcal{R}_L$ reaches the state $[u]$ and the DLW $\mathcal{A}$ reaches a state q with $L(\mathcal{A}^q) = u^{-1}L$. Indeed, otherwise we can point to a word with prefix u that is accepted only in one of the DLWs. We claim that for every state $q \in Q$ such that $L(\mathcal{A}^q) = u^{-1}L$, it holds that $\mathit{sensed}([u]) \subseteq \mathit{sensed}(q)$. To see this, consider a signal $p \in \mathit{sensed}([u])$. By definition, there exists a set $S \subseteq P$ and words u_1 and u_2 such that $([u], S\backslash\{p\}, [u_1]) \in \Delta_L$, $([u], S \cup \{p\}, [u_2]) \in \Delta_L$, yet $[u_1] \neq [u_2]$. By the definition of $\mathcal{R}_L$, there exists $z \in (2^P)^*$ such that, w.l.o.g, $z \in u_1^{-1}L \backslash u_2^{-1}L$. Hence, as $L(\mathcal{A}^q) = u^{-1}L$, we have that $\mathcal{A}^q$ accepts $(S\backslash\{p\}) \cdot z$ and rejects $(S \cup \{p\}) \cdot z$. Let $\delta_{\mathcal{A}}$ be the transition function of $\mathcal{A}$. By the above, $\delta_{\mathcal{A}}(q, S\backslash\{p\}) \neq \delta_{\mathcal{A}}(q, S \cup \{p\})$. Therefore, $p \in \mathit{sensed}(q)$, and we are done. Now, $\mathit{sensed}([u]) \subseteq \mathit{sensed}(q)$ implies that $\mathit{scost}(q) \geqslant \mathit{scost}([u])$. Since our assumption on q is only that $L(\mathcal{A}^q) = u^{-1}L$, we get that $q \in \mathit{states}(\mathcal{A}, [u])$. Thus, we conclude that for every $s \in \langle L\rangle\backslash\{[\bot]\}$ and for every $q \in \mathit{states}(\mathcal{A}, s)$ we have that $\mathit{scost}(q) \geqslant \mathit{scost}(s)$.

Next, consider the Markov chains $\mathcal{M}_{\mathcal{A}}$ and $\mathcal{M}_{\mathcal{R}_L}$, and let P and R be their respective transition matrices. We index the rows and columns of P (resp. R) by Q (resp. $\langle L\rangle\backslash\{[\bot]\}$). Let $v^0 \in [0,1]^Q$ be the initial vector for $\mathcal{A}$, thus $v^0(q_0) = 1$ and $v^0(q) = 0$ for $q \neq q_0$. Similarly, let $u^0([\epsilon]) = 1$ and $u^0([u]) = 0$ for all $[u] \neq [\epsilon]$. For $m \in \mathbb{N}$, let $v^m = v^0 P^m$ and $u^m = u^0 R^m$.

We claim that for every $s \in \langle L\rangle\backslash\{[\bot]\}$ it holds that $u^m(s) = \sum_{q \in \mathit{states}(\mathcal{A},s)} v^m(q)$.

The proof of the claim proceeds by an induction on m. For $m = 0$, we have $q_0 \in [\epsilon]$, and the claim follows trivially. Assume correctness for m, we prove the claim for $m+1$.

Consider states $s, s' \in \langle L\rangle\backslash\{[\bot]\}$. For every state $q \in \mathit{states}(\mathcal{A}, s)$ it holds that

$$R_{s,s'} = \frac{|\{\sigma : \delta_L(s,\sigma) = s'\}|}{\mathit{dom}(s)} = \frac{|\{\sigma : \delta(q,\sigma) = q' \in \mathit{states}(\mathcal{A}, s')\}|}{\mathit{dom}(q)} = \sum_{q' \in \mathit{states}(\mathcal{A},s')} P_{q,q'} \tag{1}$$

where the second equality follows from Observation 3.3 and by observing that $\mathit{dom}(q) = \mathit{dom}(s)$. The latter holds since if $\sigma \in \mathit{dom}(q)$, then there exists $q' \in Q$ such that $q' = \delta(q, \sigma)$, so $q' \in \mathit{states}(\delta_L(s, \sigma))$ and $\sigma \in \mathit{dom}(s)$, so $\mathit{dom}(q) \subseteq \mathit{dom}(s)$, and conversely - if $\sigma \in \mathit{dom}(s)$ then by Proposition 3.3 we have that $\sigma \in \mathit{dom}(q)$, so $\mathit{dom}(s) \subseteq \mathit{dom}(q)$.

Now, for every state $s' \in \langle L\rangle\backslash\{[\bot]\}$, we have that

$$\begin{aligned}
u^{m+1}(s') &= \sum_{s \in \langle L\rangle\backslash\{[\bot]\}} R_{s,s'} u^m(s) \\
&= \sum_{s \in \langle L\rangle\backslash\{[\bot]\}} R_{s,s'} \sum_{q \in \mathit{states}(\mathcal{A},s)} v^m(q) && \text{(Induction Hypothesis)} \\
&= \sum_{s \in \langle L\rangle\backslash\{[\bot]\}} \sum_{q \in \mathit{states}(\mathcal{A},s)} R_{s,s'} v^m(q) \\
&= \sum_{s \in \langle L\rangle\backslash\{[\bot]\}} \sum_{q \in \mathit{states}(\mathcal{A},s)} \sum_{q' \in \mathit{states}(\mathcal{A},s')} P_{q,q'} v^m(q) && \text{(Equation (1))} \\
&= \sum_{q' \in \mathit{states}(\mathcal{A},s')} \sum_{s \in \langle L\rangle\backslash\{[\bot]\}} \sum_{q \in \mathit{states}(\mathcal{A},s)} P_{q,q'} v^m(q) \\
&= \sum_{q' \in \mathit{states}(\mathcal{A},s')} \sum_{q \in Q} P_{q,q'} v^m(q) && \text{(Proposition 3.3)} \\
&= \sum_{q' \in \mathit{states}(\mathcal{A},s')} v^{m+1}(q')
\end{aligned}$$

and the induction is complete.

Now, let π and τ be the limiting distributions of $\mathcal{M}_{\mathcal{A}}$ and $\mathcal{M}_{\mathcal{R}_L}$ respectively, then $\pi = \lim_{n\to\infty} \frac{1}{n}\sum_{m=1}^{n} v^0 P^n$ and $\tau = \lim_{n\to\infty} \frac{1}{n}\sum_{m=1}^{n} u^0 R^n$, and by the above we have that $\tau(s) = \sum_{q\in states(\mathcal{A},s)} \pi(q)$.

Since $scost(q) \geqslant scost(s)$ for every $q \in states(\mathcal{A}, s)$, we conclude that

$$\begin{aligned}
lcost(\mathcal{A}) &= \sum_{q\in Q} \pi(q) scost(q) = \sum_{s\in\langle L\rangle\setminus\{[\bot]\}} \sum_{q\in states(\mathcal{A},s)} \pi(q) scost(q) \\
&\geqslant \sum_{s\in\langle L\rangle\setminus\{[\bot]\}} \sum_{q\in states(\mathcal{A},s)} \pi(q) scost(s) = \sum_{s\in\langle L\rangle\setminus\{[\bot]\}} scost(s) \sum_{q\in states(\mathcal{A},s)} \pi(q) \\
&= \sum_{s\in\langle L\rangle\setminus\{[\bot]\}} scost(s)\tau(s) = lcost(\mathcal{R}_L).
\end{aligned}$$

We proceed to $wcost$. Following similar arguments as above, we see that for every finite word w, we have $scost_{\mathcal{A}}(w) \geqslant scost_{\mathcal{R}_L}(w)$. Therefore, for any $n \geqslant 0$ we have $\sum_{w\in pref(L)\cap\Sigma^n} scost_{\mathcal{A}}(w) \geqslant \sum_{w\in pref(L)\cap\Sigma^n} scost_{\mathcal{R}_L}(w)$, and finally $wcost(\mathcal{A}) \geqslant wcost(\mathcal{R}_L)$. This shows that $wcost(\mathcal{R}_L) = wcost(L)$.

A.3 Computing $wcost(\mathcal{A})$ in Example 3.2

First, note that $\neg a \vee \neg b$ corresponds to 3 letters, $a \wedge b$ to 1 letter, and a to 2 letters.

We have $C_{q_0,q_0}(z) = \frac{1}{1-3z}$, $C_{q_1,q_1}(z) = \frac{1}{1-2z}$, and $C_{q_0,q_1}(z) = C_{q_0,q_0}(z)C_{q_1,q_1}(z) = \frac{1}{(1-3z)(1-2z)}$, whereas $C_{q_1,q_0}(z) = 0$.

Moreover, we have

$$\begin{aligned}
S(z) &= C_{q_0,q_0}(z) + C_{q_0,q_1}(z) = \frac{2-5z}{(1-3z)(1-2z)} \\
F_{q_0}(z) &= C_{q_0,q_0}(z)(C_{q_0,q_0}(z) + C_{q_0,q_1}(z)) = \frac{2-5z}{(1-3z)^2(1-2z)} \\
F_{q_1}(z) &= C_{q_0,q_1}(z)C_{q_1,q_1}(z) = \frac{1}{(1-3z)(1-2z)}
\end{aligned}$$

Using standard algorithms on rational functions, we get $[z^n]S(z) \sim \frac{2-5/3}{1-2/3}3^n = 3^n$, $[z^n]F_{q_0}(z) \sim n3^n$ and $[z^n]F_{q_1}(z) \sim 3^{n+1}$. We finally obtain $wcost(\mathcal{A}) = \lim_{n\to\infty} \frac{2\cdot n3^n + 1\cdot 3^{n+1}}{n3^n} = 2$.

Note that $wcost$, unlike $scost$ and $lcost$, allows to take into a consideration the cost of transient components when a long word in L is likely to spend time in them.

If the self-loop on q_0 has been labeled by two letters, say by b, rather than by three, then q_0 and q_1 would have participated equally and we would have gotten $wcost(\mathcal{A}) = 3/2$.

A.4 Proof of Theorem 4.1

We claim that $cost(\mathcal{M}) = scost_{I,O}(\mathcal{A})$, and that a minimal-cost policy f in $\mathcal{M}$ induces a minimally-sensing I/O-transducer that realizes $\mathcal{A}$.

Consider a memoryless minimal-cost policy f. We construct from f a transducer $\mathcal{T} = \langle I, O, \mathcal{S}, \text{START}, \mu, \rho \rangle$, where μ and ρ are defined as follows. For sets $i_1 \subseteq i \subseteq I$, we say that an assignment $\ell : I \to \{0, 1, \perp\}$ the (unique) i-sensed i_1-true assignment if for every signal $b \in I$, we have that $\ell(b)$ is $\perp$ if $b \notin i$, is 1 if $b \in i_1$, and is 0 if $b \in i \backslash i_1$.

Let $f(\text{START}) = \langle o_0, i_0 \rangle$. We arbitrarily [4] set $\rho(\text{START})$ to o_0. For input $i \in 2^I$, we set $\mu(\text{START}, i) = \langle \{q_0\}, \ell_0 \rangle$, for the i_0-sensed i-true assignment ℓ_0.

Next, consider a state $\langle S, \ell \rangle$ and an input $i \in 2^I$. Let $\langle o, i' \rangle = f(\langle S, \ell \rangle)$. We define $\rho(\langle S, \ell \rangle) = o$ and $\mu(\langle S, \ell \rangle, i) = \langle S', \ell' \rangle$, where $S' = \bigcup_{q \in S} \bigcup_{i_1 \subseteq \ell_\perp} \delta(q, \ell_1 \cup i_1 \cup o)$ and ℓ' is the i'-sensed i-true assignment.

We claim that $scost(\mathcal{T}) = cost(f)$. To see this, let $\mathcal{M}'$ be the Markov Chain obtained from $\mathcal{M}$ by fixing the action in each state according to f, and let $\mathcal{T}'$ be the Markov chain obtained from $\mathcal{T}$ by assigning uniform distributions to the input signals. It is easy to see that the Markov chains $\mathcal{M}'$ and $\mathcal{T}'$ are identical (with the exception of START, which, as we mentioned, does not affect the cost). Thus, $cost(f) = scost(\mathcal{T})$. Since $cost(\mathcal{M}) = cost(f)$, we can conclude that there is a transducer $\mathcal{T}$ that realizes $\mathcal{A}$ and for which $scost(\mathcal{T}) = cost(\mathcal{M})$. Thus, $scost_{I,O}(\mathcal{A}) \leqslant cost(\mathcal{M})$.

Conversely, consider a transducer $\mathcal{T}$ for $\mathcal{A}$. By following the set of sensed input signals and the output at each state, $\mathcal{T}$ induces a (possibly non-memoryless) policy f in $\mathcal{M}$. Moreover, as above, $cost(f) = scost(\mathcal{T})$. Thus, $scost(\mathcal{T}) \geqslant cost(\mathcal{M})$. Since this holds for all transducers $\mathcal{T}$, it follows that $scost_{I,O}(\mathcal{A}) \geqslant cost(\mathcal{M})$.

A.5 Proof of Theorem 4.2

We define $\mathcal{A}_k = \{2^{I \cup O}, Q, q_{\text{reset}}, \delta\}$, where

- $I = \{i_1, ..., i_{\lceil \log k \rceil}\} \cup \{dI\}$, and we view 2^I as $\{1, ..., k\} \times \{dI, \neg dI\}$. Then, $O = \bigcup_{i=1}^{k} \{o_{i,1}, ..., o_{i,\lceil \log p_i \rceil}\} \cup \{dO, e\}$, and we view it as $(\bigtimes_{i=1}^{k} \{0, ..., p_i - 1\}) \times \{dO, \neg dO\} \times \{e, \neg e\}$. Thus, a letter $\sigma \in 2^{I \cup O}$ is a pair $\sigma = \langle \sigma_I, \sigma_O \rangle$ with $\sigma_I = \langle m, dI \rangle$ where $1 \leqslant m \leqslant k$ and $dI \in \{0, 1\}$, and with $\sigma_O = \langle r_1, ..., r_k, dO, e \rangle$ with $0 \leqslant r_i < p_i - 1$ for all $1 \leqslant i \leqslant k$, $dO \in \{0, 1\}$, and $e \in \{0, 1\}$.

- $Q = \{q_{\text{reset}}\} \cup \bigcup_{i=1}^{k} \{q_{i,0}, \ldots, q_{i,p_i - 1}\}$.

- The transition function δ is defined as follows. Consider a letter $\sigma = \langle \langle m, dI \rangle, \langle r_1, ..., r_k, dO, e \rangle \rangle$. In state q_{reset}, we have $\delta(q_{\text{reset}}, \sigma) = q_{m,0}$. For a state $q_{i,j}$, we have

$$\delta(q_{i,j}, \sigma) = \begin{cases} q_{i,(j+1) \bmod p_i} & e = 0 \wedge r_i = j \wedge dI = dO \\ q_{\text{reset}} & j = 0 \wedge e = 1 \wedge r_i = j \wedge dI = dO \end{cases}$$

  Note that $\delta(q_{i,j}, \sigma)$ is undefined in all other cases, thus when $dI \neq dO$, when $j \neq 0$ and $e = 1$, and when $r_i \neq j$.

[4] Since the output in START is ignored, this is indeed arbitrary.

Intuitively, in order to take a transition from $q_{i,j}$, r_i has to match j, and $dI = dO$. Providing that, if $e = 0$ then the transition progresses along the cycle in the i component, and if $e = 1$ and the state is $q_{i,0}$, then the run moves to q_{reset}.

Consider a transducer $\mathcal{T}$ with $scost(\mathcal{T}) < 1$ that realizes $\mathcal{A}_k$. We show that $\mathcal{T}$ must have at least $\prod_{i=1}^{k} p_i$ states. Observe that whenever $\mathcal{A}_k$ is in a state that is different from q_{reset}, the transducer $\mathcal{T}$ must sense at least dI in order to match dO. Thus, the only state that $\mathcal{A}_k$ visits and in which the sensing cost can be lower than 1 (i.e. 0) is q_{reset}. Thus, $\mathcal{T}$ senses 0 in q_{reset}, and moreover, $\mathcal{A}_k$ has to visit q_{reset} infinitely often in order for the 0 sensing to reduce the sensing cost. Sensing 0 in q_{reset} means that in the next step, $\mathcal{A}_k$ can be in $q_{i,0}$ for every $1 \leqslant i \leqslant k$. Since $\mathcal{T}$ has to output $r_i = j$ in every $q_{i,j}$, then $\mathcal{T}$ has to keep track of j, which runs from 0 to $p_i - 1$. Since $\mathcal{T}$ does not "know" the value of i, then $\mathcal{T}$ has to keep track of every reachable combination of $r_1, ..., r_k$ with $0 \leqslant r_i \leqslant p_i - 1$ for every $1 \leqslant i \leqslant k$. From the Chinese remainder theorem, every such possible combination of $r_1, ..., r_k$ is reachable. Thus, $\mathcal{T}$ needs at least $\prod_{i=1}^{k} p_i$ states.

Finally, note that that there is a transducer $\mathcal{T}$ that realizes $\mathcal{A}_k$. Indeed, $\mathcal{T}$ has $\prod_{i=1}^{k} p_i$ states and outputs e, thus causing $\mathcal{A}_k$ to return to q_{reset}, every $\prod_{i=1}^{k} p_i$ steps.

B Non-uniform Distributions

In this section we consider non-uniform distributions, and show that incorporating them in the model does not involve any significant changes in our techniques. We start by defining the model.

In the closed setting, we are given a *labeled Markov chain* $\mathcal{M} = \langle S, P, \ell \rangle$ where ℓ is a labeling function $\ell : S \to \Sigma$. The probability of a letter $l \in \Sigma$ in state s is $P_s(l) = \sum_{s' \in S : \ell(s') = l} P(s, s')$. We assume that the probability has full support. That is, for every $s \in S$ and $l \in \Sigma$ it holds that $P_s(l) > 0$, and that all probabilities are rational. We lift the probability to words as follows. A sequence of states $\pi = s_1, ..., s_n$ induces the word $w_\pi = \ell(s_1) \cdots \ell(s_n)$. The probability of a word w from a state s is then $P_s(w) = \sum_{\pi : w = w_\pi} P_s(\pi)$, where $P_s(\pi) = \prod_{1 \leqslant i \leqslant n} P(s_{i-1}, s_i)$, where we set $s_0 = s$ and $\pi = s_1, \ldots, s_n$. The distribution on Σ^* is lifted to a distribution on Σ^ω based on cylinder sets, in the usual manner.

In the open setting, the specification is over $2^{I \cup O}$. We are given an MDP $\mathcal{M} = \langle S, s_0, (A_s)_{s \in S}, \mathrm{P}, cost \rangle$ and a labeling function $\ell : S \to 2^{AP}$, where $A_s = 2^O$ for every $s \in S$. Then, for a sequence of outputs $\rho \in (2^O)^*$ (which corresponds to a strategy for the MDP), the induced Markov chain defined the probability space on $(2^I)^*$, similarly to the above.

In the following we explain how to adapt the various results of the paper to the setting of non-uniform inputs.

B.1 The letter based approach - adapting Theorem 3.1

In the letter based approach, we still sample letters, restricting to those that keep us within the language. However, instead of sampling them uniformly, we need to take into account the MC describing the distribution. Thus, we are given a DLW $\mathcal{A} = \langle \Sigma, Q, \delta, q_0 \rangle$ and a labeled MC $\mathcal{M} = \langle S, P, \ell \rangle$ as above. Instead of constructing $\mathcal{M}_{\mathcal{A}}$ as in Section 3, we construct $\mathcal{M}_{\mathcal{A}}$ as follows. We start with the product of $\mathcal{A}$ and $\mathcal{M}$. That is, the state space is $Q \times S$, there is a transition between $\langle q, s \rangle$ and $\langle q', s' \rangle$ if $\delta(q, \sigma) = q'$ in $\mathcal{A}$, and the probability of the transition is determined according to $\mathcal{M}$. Since δ is only a partial function, we re-normalize the transition probabilities. Computing the sensing cost $lcost(\mathcal{A})$ under $\mathcal{M}$ proceeds by finding a limiting distribution in this chain, similarly to Theorem 3.1.

B.2 The letter based approach - adapting Theorem 3.2

Computing $wcost(\mathcal{A})$ is done similarly to the case of a uniform distribution, with the difference being the construction of the generating function $C_{p,q}(z)$. Instead of having the coefficient of z^n being $\#paths(p, q, n)$, we need to account for the probabilities of the different paths. To do so, we consider again the product of $\mathcal{A}$ and the states S of the MC describing the distribution. Consider a state $\langle q, s \rangle$. Since the transition probabilities in $Q \times S$ are assumed to be rational, we assume that these probabilities are all multiples of some common denominator p. Then, when computing the number of paths from $\langle p, s \rangle$ to $\langle q, s' \rangle$, we count each path according to its multiple of p. The rest of the computation is the same as the proof of Theorem 3.2.

B.3 The open setting - adapting Theorem 4.1

Consider a realizable DLW specification $\mathcal{A}$ over $2^{I \cup O}$, and an MDP $\mathcal{M} = \langle S, s_0, (A_s)_{s \in S}, \mathrm{P}, cost \rangle$ describing the input distribution. Instead of directly constructing an MDP from $\mathcal{A}$ (using a uniform distribution), we construct the MDP from $\mathcal{A}$ using the probabilities defined by $\mathcal{M}$, as follows. The state space of the MDP is $\mathcal{S} = (2^Q \times \{0, 1, \perp\}^I \times S) \cup \{\textsc{start}\}$. Next, Consider a state $\langle R, \ell, s \rangle$ and an action $\langle o, i \rangle$. For an available transition to state $\langle R', \ell', s' \rangle$ the transition probability is defined as the probability to read ℓ'_1 in s'. That is, $P_s(\ell'_1)$. The rest of the proof follows the same lines as that of Theorem 4.1. We note that the size of a memoryless strategy may now depend (polynomially) on the size of the distribution MDP.

6 Discussion

Aftermath of the Work

We introduced and studied a range of quantitative extensions to formal methods, aimed at formalizing the quality of systems and solving the relevant problems.

We showed that quantitativeness may stem from different sources: from the specification alone, as in LTL$[\mathcal{F}]$ and LTL$^{\text{disc}}[\mathcal{D}]$, where we specify the quality of a Boolean system using a quantitative specification formalism, from the system alone, as with sensing costs, where every system is associated with a sensing cost, and we look for a minimal-sensing system that satisfies a Boolean specification, and from both sources together, as in latticed-LTL, where the formula, the system, and the satisfaction value are multi-valued. Reasoning about the different extensions naturally requires various mathematical tools, and gives rise to different challenges, both technical and philosophical. We elaborate on some of these.

The propositional-quality logic LTL$[\mathcal{F}]$ allows arbitrary relations between subformulas, which is as much as one could hope for in the propositional level. Still, we are able to show that algorithms for LTL$[\mathcal{F}]$ match the complexity of their Boolean analogues. This is surprisingly good news, even more so once we showed that LTL$[\mathcal{F}]$ allows for succinct representations of LTL formulas, thus even benefiting the Boolean setting. However, as we show, LTL$[\mathcal{F}]$ does not actually add expressive power to LTL, which means that in practice, its use is mainly for succinct representation, and for ease of specifying quantitative properties. This also explains why we managed to adapt Boolean methods in a clean way to tackle problems for LTL$[\mathcal{F}]$.

Contrary to LTL$[\mathcal{F}]$, the temporal-quality logic LTL$^{\text{disc}}[\mathcal{D}]$ admits properties that are not ω-regular. This alone makes algorithms for LTL$^{\text{disc}}[\mathcal{D}]$ much more challenging. Indeed, as we saw, our constructions for LTL$^{\text{disc}}[\mathcal{D}]$ are much less straightforward than those of LTL$[\mathcal{F}]$, and in fact make use of some "heavy duty" mathematical tools, such as succinct manipulation of arithmetic circuits.

Interestingly, while LTL$[\mathcal{F}]$ alone does not add expressive power, the combination of LTL$[\mathcal{F}]$ with LTL$^{\text{disc}}[\mathcal{D}]$ adds too much expressive power, rendering most problems undecidable.

We proceeded to introduce LLTL, a multi-valued formalism that is evaluated over multi-valued systems. We saw that LLTL allows us to reason about properties such as inconsistent viewpoints. Technically, the definition of LLTL is very clean, as LTL readily evaluates over (de-Morgan) lattices, with the latter lending themselves elegantly to the syntax of LTL. However, since lattices are more intricate than the Boolean setting, we did get interesting phenomena that does not occur in the Boolean setting. For example, the value of the LLTL formula $\mathsf{F}p$ in a computation may not be a value that is assigned to

p at any single point in the computation.

These intricacies also appeared in the translation of LLTL to Boolean automata, both in a more elaborate construction than the Boolean setting, or even that of $\text{LTL}[\mathcal{F}]$, as well as in the correctness proof, which crucially relies on Birkhoff's representation theorem.

Finally, we considered sensing cost – a new complexity measure for systems. There, quality is a property of the system alone. Traditional (Boolean) formal methods typically concern worst-case analysis (e.g., a system satisfies a specification if every computation of the system satisfies the specification). This was also the case for $\text{LTL}[\mathcal{F}]$, $\text{LTL}^{\text{disc}}[\mathcal{D}]$, and LLTL. In sensing, however, we keep the Boolean approach, and measure sensing only in expectation (under a probability space). In this view of quality, we do not refine the notion of correctness, but rather put an additional measure on systems that are already correct.

Technically, the introduction of probability renders the setting much more involved mathematically. Fortunately, it also brings a new arsenal of available tools, ranging from Markov chains to analytic combinatorics, which we use in order to tackle the problems at hand.

Directions for Future Research

The thesis lays the ground for several quantitative extensions of Boolean formal methods. For most of the problems we tackle, we provide complete solutions (e.g., tight complexity bounds), yet there are many directions for future research.

The most immediate directions are closing the gaps on problems that are left open in this thesis: In Chapter 2, the complexity of $\text{LTL}^{\text{disc}}[\mathcal{D}]$ model-checking for general discounting functions, as well as solving the case of non-strict inequality in the satisfiability problem for $\text{LTL}^{\text{disc}}[\mathcal{D}]$, are left open. In Chapter 5, we only consider synthesis of safety properties. In the general case, we are given a specification by means of a DPW over assignments to $I \cup O$, and the goal is to compute the minimal sensing cost of a system that realizes the specification. The latter was recently addressed and solved, but was not yet published at the time of writing the thesis.

Another important direction for future research involves the combination and trade-off of different quality measures. Perhaps the most obvious example is the trade-off between the waiting time for fulfillment of eventualities and the sensing cost. Indeed, as we show in Chapter 4, the sensing cost of ω-regular languages may sometimes be attained only as a limit of a sequence of automata. Specifically, the construction of these automata involves lazy sensing. Crucially, this increases the intervals between consecutive visits to accepting states. This corresponds to lowering the quality of the system, if discounting is concerned. It is thus interesting to find the Pareto curve of such combined, conflicting, requirements.

Other conflicting combinations that are of interest include minimizing the number of states and the sensing cost simultaneously, or minimizing the number of states while increasing the quality (which may or may not be conflicting, depending on the specification).

המפרט בערך הסיפוק הרצוי. אנו חוקרים את בעיית הסינתזה המורעשת ל LLTL, וכן אספקטים תיאורטיים נוספים של סביבה זו, למשל את כמות הרעש שמשרן מסוגל להתמודד עמה, ואת ההשפעה של הרעשת הקלט על ערך הסיפוק של הנוסחה.

שתי התרומות האחרונות בעבודה עוסקות בפן אחר של איכות, בו מדד האיכות נובע ישירות מהמערכת. אנו מציגים וחוקרים מדד סיבוכיות חדש עבור שפות רגולריות, הנקרא *עלות חישה* (sensing cost). באופן אינטואיטיבי, עלות החישה של שפה היא מדד לרמת הפירוט בו עלינו לקרוא מילת קלט אקראית על מנת להכריע את שייכותה לשפה.

אנו מראים שעבור מלים סופיות, מזעור מספר המצבים של אוטומט עשוי רק להקטין את עלות החישה שלו. בפרט, מאחר ואוטומטים דטרמיניסטיים מעל מלים סופיות (DFA) ניתנים למזעור בזמן פולינומי, אנו יכולים לבנות בזמן פולינומי אוטומט בעל עלות חישה מינימלית, וכן לחשב בזמן פולינומי את עלות החישה של שפה הנתונה על ידי DFA.

אנו ממשיכים לחקר שפות אומגה-רגולריות, הנתונות על ידי אוטומטי זוגיות דטרמיניסטיים (DPA), ומראים שבניגוד למלים סופיות, חישה מינימלית עשויה להתקבל רק כגבול של סדרת אוטומטים אינסופית. אנו מראים שהחישה המינימלית בגבול כזה ניתנת לאפיון על ידי מחלקות השקילות הימניות של השפה, מה שמאפשר לנו לחשב בזמן פולינומי את עלות החישה המינימלית של שפות אומגה-רגולריות.

בהמשך, אנו מתמקדים בניטור (monitoring). במקרה זה, אנו מתעניינים רק בעלות החישה של מלים הנמצאות בשפה. לשם כך, אנו מפתחים תשתיות שונות וחדשות לטיפול במרחב המדגם של מלים בשפה. אנו מראים שעבור ניטור, החישה המינימלית מתקבלת במשגוח (monitor) המבוסס על אוטומט דטרמיניסטי מינימלי עבור השפה.

לבסוף, אנו חוקרים את עלות החישה בהקשר של סינתזה. אנו מראים שבמקרה זה צצים אתגרים חדשים: מזעור עלות החישה של מפרט עשוי לדרוש משרנים בגודל אקספוננציאלי, ובעית הסינתזה של משרן בעל חישה מינימלית היא שלמה ב EXPTIME, אפילו עבור מפרטי בטיחות (safety), הניתנים כאוטומטים דטרמיניסטיים.

תקציר

שיטות פורמליות מסורתיות עוסקות בשתי בעיות עיקריות: *בדיקת מודל*, ו*סינתזה*. בבדיקת מודל, אנו מקבלים מערכת ומפרט, ועלינו לבדוק האם המערכת מספקת את המפרט. בסינתזה, אנו מקבלים מפרט בלבד, ועלינו לבנות, בצורה אוטומטית, מערכת נכונה - כזו המספקת את המפרט.

נכונות היא בוליאנית: מערכת מספקת, או לא מספקת, מפרט נתון. אולם, עושרן של המערכות בימינו מצדיק פורמליזם כמותי.

בעבודה זו, אנו מכלילים את בדיקת המודל והסינתזה לסביבה הכמותית, וחוקרים אותן, ובעיות נוספות העולות בסביבה זו. בהכללה זו, אנו מתעניינים לא רק בשאלה *האם* המערכת מספקת את המפרט, אלא גם *באיזו איכות* המערכת מספקת את המפרט.

תחילה, אנו מציגים שפות מפרט אשר מאפשרות למתכנן המערכת לנסח באופן פורמלי את איכות המערכת, בעזרת *קשרי איכות*.

אנו מבדילים בין שתי גישות לניסוח איכות. הראשונה, *איכות פסוקית* (propositional quality), מוסיפה לשפת המפרט הבוליאנית קשרי-איכות פסוקיים, אשר מאפשרים לתעדף ולמשקל אפשרויות סיפוק שונות למפרט. השניה, *איכות עתית* (temporal quality), מעדנת את הקשר העתי "בסופו של דבר" (eventually) בעזרת קשרי היוון (discounting), שערכם מערב את פרק הזמן שעבר עד סיפוקם.

אנו מציגים שתי הכללות כמותיות ללוגיקה העתית הלינארית (LTL), האחת על ידי קשרי איכות פסוקיים, והשניה על ידי קשרי היוון. בשתי ההכללות, ערך הסיפוק של מפרט הוא מספר בקטע [0,1], אשר מתאר את איכות הסיפוק. אנו מדגימים את שימושיות ההכללות וחוקרים את כריעות ואת סיבוכיות בעיות ההכרעה והחיפוש עבורן, ועבור הכללה של LTL המשלבת את שני סוגי הקשרים.

תרומתנו הבאה היא הכללה של LTL לתחום מרובה-ערכים, באופן ספציפי – סריג (lattice). *בלוגיקה עתית מסורגת* (LLTL), פעולות הגימום (and) והאיווי (or) מזוהות בהתאמה עם פעולות המפגש (meet) והמצרף (join) של הסריג L. נוסחת LLTL מעל הפסוקים האטומיים AP משוערכת על חישובי-סריג, כלומר חישובים ב $(AP^L)^\omega$, ולכן ערך הסיפוק של נוסחה בחישוב הוא גם כן איבר בסריג L. סביבת הסריגים עולה בצורה טבעית באופן מעשי, למשל במפרטים המערבים סדרי עדיפויות, או במערכות עם נקודות מבט לא עקביות.

אנו פותרים את בעיית הסינתזה עבור LLTL, שבה עלינו לבנות משרן (transducer) המממש את המפרט הנתון בערך סיפוק נתון.

בסינתזה בוליאנית, חוקרים עסקו בסביבות עם מידע חלקי, בהן ערכי האמת של חלק מהקלטים נסתרים, ובכל זאת עלינו לבנות משרן המממש מפרט נתון. בסביבה מרובת ערכים, אנו מציגים וחוקרים סוג חדש של מידע חלקי, בו ערכי האמת של חלק מהקלטים עשויים להיות *מורעשים*, ולמרות זאת על המשרן לממש את

עבודה זו נעשתה בהדרכתה של אורנה קופרמן.

על הצרנה, הסקה, ואיכות

חיבור לשם קבלת תואר דוקטור לפילוסופיה

מאת

שאול אלמגור

הוגש לסנט האוניברסיטה העברית בירושלים

מאי 2016