A Monitoring-Oriented Theory and Classification of Quantitative Specifications

by

N. Ege Saraç

June, 2025

A thesis submitted to the
Graduate School
of the
Institute of Science and Technology Austria
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

Committee in charge:

Beatriz Vicoso, Chair
Thomas A. Henzinger
Krishnendu Chatterjee
Bernd Finkbeiner
Dejan Ničković

The thesis of N. Ege Saraç, titled *A Monitoring-Oriented Theory and Classification of Quantitative Specifications*, is approved by:

Supervisor: Thomas A. Henzinger, ISTA, Klosterneuburg, Austria

Signature: _____

Committee Member: Krishnendu Chatterjee, ISTA, Klosterneuburg, Austria

Signature: _____

Committee Member: Bernd Finkbeiner, CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Signature: _____

Committee Member: Dejan Ničković, AIT Austrian Institute of Technology, Vienna, Austria

Signature: _____

Defense Chair: Beatriz Vicoso, ISTA, Klosterneuburg, Austria

Signature: _____

Signed page is on file

ISTA Thesis, ISSN: 2663-337X

I hereby declare that this thesis is my own work and that it does not contain other people's work without this being so stated; this thesis does not contain my previous work without this being stated, and the bibliography contains all the literature that I used in writing the dissertation.

I accept full responsibility for the content and factual accuracy of this work, including the data and their analysis and presentation, and the text and citation of other work.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee, and that this thesis has not been submitted for a higher degree to any other university or institution.

I certify that any republication of materials presented in this thesis has been approved by the relevant publishers and co-authors.

Signature: _____

N. Ege Saraç
June, 2025

Signed page is on file

Abstract

Quantitative properties offer a framework for specifying and verifying system behaviors beyond the traditional boolean perspective. For example, while a boolean property may specify whether a server eventually grants every request it receives, a quantitative one may map each server execution to its average response time. This quantitative view is relatively well-studied in the context of static verification. However, although such properties often appear in practice as performance or robustness measures in a dynamic verification context, a general theoretical framework for their analysis and classification from a monitoring perspective is still missing.

In this thesis, we aim to develop such a framework that takes resource-precision tradeoffs of monitors as a central consideration. We present the first theory of monitorability for quantitative properties where monitors can be naturally approximate and compared regarding their precision and resource use. In particular, we show that additional monitor resources such as registers or states lead to strictly better approximations for some properties. To enable such analyses in a machine-model independent way, we describe an abstract notion of monitors that can be instantiated with concrete models of monitors. Within this framework, we study how abstract monitors behave and identify classes of properties amenable to approximate monitoring with resource-precision considerations. We then extend the boolean safety-liveness dichotomy and safety-progress hierarchy to the quantitative setting with a monitoring perspective. In particular, we prove that every property is the pointwise minimum of a safety property and a liveness property, and properties that are both safe and co-safe can be approximately monitored arbitrarily precisely using only finitely many states. We also study the classes of quantitative properties definable by finite-state quantitative automata and provide algorithms for deciding their safety or liveness as well as their safety-liveness decompositions. Finally, we present the first general-purpose tool for automating the analysis, verification, and monitoring of quantitative automata.

Acknowledgements

First and foremost, I owe my deepest thanks to my supervisor, Tom Henzinger. His balance of uncompromising standards and continuous encouragement has both challenged me to strive for excellence and given me the confidence to pursue ambitious goals. Every discussion with him has been filled with inspiration and clarity—I have yet to leave one without feeling energized and excited. I am genuinely grateful for his guidance; it has been a privilege to be his student.

I am equally indebted to the members of my thesis committee, Krishnendu Chatterjee, Bernd Finkbeiner, and Dejan Ničković, whose insight and feedback over the years have profoundly shaped my growth as a researcher. Our conversations broadened my perspective and encouraged me to explore questions I had not previously considered. I feel fortunate to have learned from such remarkable mentors. I also appreciate Beatriz Vicoso for chairing both my qualifying exam and my thesis defense.

My heartfelt thanks also go to all my collaborators. Their expertise, creativity, and collegial spirit have made a significant impact on my work. I am especially grateful to Nicolas Mazzocchi for his warm friendship and our fruitful collaboration, which has been the cornerstone of this thesis. In addition, I thank Guy Avni for his mentorship during my rotation in Tom's group and beyond; Udi Boker for his meticulous and upbeat approach to work; Marek Chalupa for his reliable help in translating our ideas into software; Thomas Ferrère for his patience during my internship; and Pavol Kebis for his enthusiasm and clear thinking. I also extend my gratitude to everyone in Tom's and Krish's groups, as well as my fellow PhD students at ISTA, for fostering such a supportive and inspiring community.

Finally, I thank my family and closest companions—especially my parents, my sister, and my partner—for their unwavering love and support, which have made every difficult step of this journey lighter. I dedicate this thesis to them.

Funding sources. This work was supported in part by the Austrian Science Fund (FWF) under grant Z211-N23 (Wittgenstein Award) and the ERC-2020-AdG 101020093.

About the Author

Ege Saraç received a BSc in Computer Science and Engineering with a minor in Mathematics from Sabancı University before joining ISTA in September 2019 to pursue his PhD under the supervision of Tom Henzinger. His primary research interest lies in developing and applying mathematical methods to increase trust in software systems. In particular, he has developed techniques and tools for the formal classification, verification, and runtime monitoring of quantitative system properties, enabling resource-aware approximate analyses of system performance and correctness. His work has been published in leading venues in theoretical computer science (LICS, ICALP, FoSSaCS) and formal methods (TACAS, CONCUR).

List of Collaborators and Publications

In Chapter 3, the following publications were re-used in full:

1. [HS21]: Thomas A. Henzinger, N. Ege Saraç. *Quantitative and Approximate Monitoring*. In 36th Annual ACM/IEEE Symposium on Logic in Computer Science, **LICS 2021**.[1]

In Chapter 4, the following publications were re-used in full:

2. [HMS22]: Thomas A. Henzinger, Nicolas Mazzocchi, N. Ege Saraç. *Abstract Monitors for Quantitative Specifications*. In Runtime Verification - 22nd International Conference, **RV 2022**.

In Chapter 5, the following publications were re-used in full:

3. [HMS23]: Thomas A. Henzinger, Nicolas Mazzocchi, N. Ege Saraç. *Quantitative Safety and Liveness*. In Foundations of Software Science and Computation Structures - 26th International Conference, **FoSSaCS 2023**.
4. [BHMS23]: Udi Boker, Thomas A. Henzinger, Nicolas Mazzocchi, N. Ege Saraç. *Safety and Liveness of Quantitative Automata*. In 34th International Conference on Concurrency Theory, **CONCUR 2023**.
5. [BHMS25]: Udi Boker, Thomas A. Henzinger, Nicolas Mazzocchi, N. Ege Saraç. *Safety and Liveness of Quantitative Properties and Automata*. In Logical Methods in Computer Science, **LMCS Volume 21 Issue 2 (2025)**.

In Chapter 6, the following publications were re-used in full:

6. [CHMS24]: Marek Chalupa, Thomas A. Henzinger, Nicolas Mazzocchi, N. Ege Saraç. *QuAK: Quantitative Automata Kit*. In Leveraging Applications of Formal Methods, Verification and Validation. Software Engineering Methodologies - 12th International Symposium, **ISoLA 2024**.
7. [CHMS25]: Marek Chalupa, Thomas A. Henzinger, Nicolas Mazzocchi, N. Ege Saraç. *Automating the Analysis of Quantitative Automata with QuAK*. In Tools and Algorithms for the Construction and Analysis of Systems - 31st International Conference, **TACAS 2025**.

[1] © 2021 IEEE. Reprinted, with permission, from Thomas A. Henzinger and N. Ege Saraç, "Quantitative and Approximate Monitoring," 2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), June 2021.

The works listed below resulted from or are published during the author's time at ISTA, but they are not included in the main body of the thesis.

8. [FHS18]: Thomas Ferrère, Thomas A. Henzinger, N. Ege Saraç. *A Theory of Register Monitors*. In 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, **LICS 2018**.

9. [HS20]: Thomas A. Henzinger, N. Ege Saraç. *Monitorability Under Assumptions*. In Runtime Verification - 20th International Conference, **RV 2020**.

10. [SAA+21]: N. Ege Saraç, Ömer Faruk Altun, Kamil Tolga Atam, Sertaç Karahoda, Kamer Kaya, Hüsnü Yenigün. *Boosting Expensive Synchronizing Heuristics*. In Expert Systems with Applications, **ESWA Volume 167 (2021)**.

11. [HKMS23]: Thomas A. Henzinger, Pavol Kebis, Nicolas Mazzocchi, N. Ege Saraç. *Regular Methods for Operator Precedence Languages*. In 50th International Colloquium on Automata, Languages, and Programming, **ICALP 2023**.

12. [HMS24]: Thomas A. Henzinger, Nicolas Mazzocchi, N. Ege Saraç. *Strategic Dominance: A New Preorder for Nondeterministic Processes*. In 35th International Conference on Concurrency Theory, **CONCUR 2024**.

13. [BMNS24]: Borzoo Bonakdarpour, Anik Momtaz, Dejan Ničković, N. Ege Saraç. *Approximate Distributed Monitoring Under Partial Synchrony: Balancing Speed & Accuracy*. In Runtime Verification - 24th International Conference, **RV 2024**.

14. [HKMS25]: Thomas A. Henzinger, Pavol Kebis, Nicolas Mazzocchi, N. Ege Saraç. *Quantitative Language Automata*. In 36th International Conference on Concurrency Theory, **CONCUR 2025**.

Table of Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

1.1 Motivation

Modern software systems. From our pockets to critical infrastructure, software systems are everywhere in today's world. On top of this all-encompassing presence, software systems have grown increasingly complex over the decades. Contemporary software, in particular, typically consists of large code bases, interacts extensively with other software, and evolves continuously. Furthermore, the past decade has witnessed a paradigm shift toward incorporating more and more learned components into these systems. Today, there is even a trend toward building the remaining non-learned parts of such systems *by* learned systems. These developments together lead to complexity growing at an unprecedented rate and raise questions about how we can trust such complex systems.

Static verification. The problem of trust in software systems is not new; formal methods for enabling trust have been a research focus for decades [McC60, Lan64, Flo67, Hoa69, SS71, Rey72, Dij76, CC77, SC82, HT87, JGS93, Cla97]. Static verification methods analyze a program's code or mathematical model without executing it. Crucially, these methods aim to shift trust from programmers' abilities to formal proofs about programs, leaving no room for doubt once such proofs are obtained. Developments in this area have resulted in numerous success stories, not only in academic research but also in practical applications [LPY97, Lam02, CCG+02, KNP11, DJKV17, DMB08, BBB+22, Tea24, DMKA+15].

Dynamic verification. At the other end of the spectrum, we have dynamic verification methods that analyze a program's behavior during its execution. Today, dynamic verification is often used interchangeably with testing, which, although effective at uncovering programming errors, lacks the ability to provide formal guarantees. After all, testing can show only the presence of bugs, not their absence. Nevertheless, various forms of testing have long been established as an efficient first step toward improving trust in software systems [DJK+99, Ber07, GLM+08, AO16]. Leading information technology companies actively employ both static and dynamic verification methods today to improve trust in the performance, reliability, and safety of their products [BBC+06, CDD+15, RCF+20, NRZ+15, BJA+21].

Challenges. The ambitious goals of formal verification face serious scalability challenges today. Although verification methods have consistently improved over the years [CGJ+00,

AHK02, McM03, BHHK03, CKL04, NOT06, DKW08, KBD+17, GMDC+18, BK08, CHV+18], the complexity of software systems has increased even faster. Moreover, even systems that are within our verification capabilities might still be unverifiable because they are closed source. This creates a gap between real-world software systems and those amenable to static verification. On the other hand, most dynamic verification methods offer no formal guarantees despite their effectiveness in finding bugs. To help close these gaps, we advocate for a compromise: lightweight and best-effort yet formal methods that sacrifice completeness to make a dent in the trust problem.

Monitoring and runtime verification. Runtime verification (a.k.a. monitoring) is a dynamic verification method in which a monitor observes a system during its execution and evaluates it against a *formal specification*. In this way, it bridges the mathematical rigor of static verification with the efficiency of testing. Runtime verification has received significant attention in both research and practice, especially for improving trust in complex, black-box, safety-critical systems [HR04a, HR04b, MN04, Don10, ALFS11, DDS17, NY20].

Specifying system behaviors. The traditional approach for specifying and verifying system behaviors has been boolean in nature: system executions are either correct or incorrect, specifications are either met or violated. Such a binary view, although proved valuable, falls short in providing a nuanced framework for evaluating system behaviors, considering both correctness and performance. In particular, this lack of granularity limits reasoning about how well or how poorly systems meet specified criteria and how systems perform regarding given performance measures, which are often central in contemporary software systems.

Quantitative properties. Instead of partitioning the set of all executions into correct behaviors and incorrect behaviors, the quantitative view considers specifications as functions from system executions to richer value domains, e.g., the set of reals [CDH10b]. Therefore, quantitative properties enable a nuanced way to specify and verify system behaviors. For example, such specifications can express absolute performance metrics like average response time or worst-case memory usage, or relative correctness measures indicating the degree of satisfaction or violation. This potential has been recognized in a wide range of applications.

- *Performance analysis:* Responsiveness and latency in web services and cloud infrastructures.
- *Resource utilization:* Memory and energy consumption in embedded software and mobile applications.
- *Quality of service:* Satisfaction degree of service-level agreements in telecommunications and computer networks.
- *Safety margins:* Proximity to safety thresholds in autonomous systems such as self-driving cars and robotics.

Opportunities. We believe that monitors can be used as third-party components offering best-effort assurances to improve trust in complex software systems, and thus help narrow the verification gap. From a theoretical perspective, runtime verification moves the burden from inclusion checking to membership checking—an easier problem. Nonetheless, although infinite-state monitors are used frequently in practice, little effort went into exploring the

theory of infinite-state monitors, including those for quantitative properties. In addition, quantitative monitors can be naturally approximate, in which case monitor precision can be traded against monitor resources. We argue that there is an opportunity to establish a framework for monitoring and classifying quantitative properties, explicitly incorporating tradeoffs between resource use and precision. Such a framework should combine theoretical rigor with effective automation for both monitoring and classification tasks.

1.2 Related Work and Thesis Goals

In this section, we present a high-level overview of the relevant literature to put the goals of this thesis into context. Detailed discussions are deferred to corresponding chapters in the main body of the thesis.

1.2.1 Specifying and Synthesizing Monitors

The formal specification of system properties and the automated synthesis of monitors from such specifications are central topics in monitoring. We can classify specifications, rather coarsely, as declarative (e.g., temporal logics) or executable (e.g., state machines). While declarative specifications are usually more compositional, executable specifications often admit more straightforward monitoring algorithms [BFFR18].

Linear temporal logic (LTL) and finite-state automata have long been the standard in system specification and verification. Naturally, they were also studied extensively in the context of monitoring, e.g., for synthesizing monitors in the form of automata from LTL specifications [GPVW95, HR01, GH01, HR02, FS04, HR04b, PZ06]. Later work also focused on more expressive temporal logics, including extensions with freeze quantifiers [DL09], counters [DLT15], first-order quantifiers [HP18] as well as time-constrained temporal operators [TR05] and richer first-order quantification [BKMZ15]. Similarly, various models of automata beyond finite-state have been also used for specifying and executing monitors, for example, models with quantified event parameters [BFH+12], registers storing data values [GDPT13], and discrete clocks [BNF13]. In the branching-time setting, a prominent line of work is by Aceto et al. [FAA+17, AAF+19b, AAF+19a, AAF+21a], which uses the μ-calculus and Hennessy-Milner logic.

An important consideration among these extensions is the handling of events where observations are associated with data values (from a potentially infinite set). Five notable approaches for tackling the monitoring problem in this context have been surveyed in [HRTZ18], and we summarize them below.

Stream-based monitoring [dSS+05, BFS+20, FFS+19, CHL+18, LSS+18, LSS+19] frames the monitoring problem as a special case of stream processing by treating the monitor inputs and outputs as data streams. These frameworks typically feature a flexible design where users write stream expressions defining the relationship between inputs and outputs. The monitoring algorithms then incrementally compute the output values based on these expressions and the previous values in the streams.

The parametric trace slicing method [AAC+05, CR09, RC12, BFH+12, JMLR12, RCR15] projects an execution trace into subtraces that correspond to specific sets of data values. Essentially, the slicing filters the original trace to consider only the events that are relevant to a particular parameter valuation. The monitor, therefore, can evaluate the behavior for each

parameter instance separately. Monitoring in this setting consists of two phases: first, slicing the trace according to data values of events, and second, checking each slice individually against the specification using standard approaches. Thanks to efficient indexing techniques, parametric trace slicing admits a relatively low monitoring overhead.

The rule-based monitoring approach [BRH08, Hav15] was inspired by rule-based production systems in artificial intelligence. In this approach, system states are treated as sets of facts, and specifications are expressed as rules of the form *conditions* $\Rightarrow$ *action*, which naturally supports processing data languages. The monitoring algorithm then consist in matching the current state against the rule conditions.

First-order temporal logics [BHKZ12, BKMZ15, BKZ17, BDH$^+$20] extend traditional temporal logics with quantification over data, allowing one to specify properties that relate the data values within events. The monitoring algorithm translates these formulas into relational algebra expressions (i.e., efficient operations such as filtering, projecting, and joining sets of tuples [GMUW08]) to enable the reuse of intermediate results when checking data-dependent properties. An alternative approach [HPU20] uses instead binary decision diagrams to represent sets of assignments, which results in a faster monitoring procedure.

Monitoring modulo theories [DLT13b, DLT16] generalizes monitoring procedures from propositional temporal logics to a setting where data constraints are taken from a specific first-order theory. Since the temporal aspect of the specification and the data aspect are decoupled, the two constraints can be treated separately: the temporal aspect is handled by established techniques from the underlying temporal logic, and the data aspect by SMT solvers implementing decision procedures for the underlying first-order theory.

Various combinations of these methods have been also considered, for example, extensions of stream-based monitoring with slicing [FFST16, GS21] as well as first-order temporal logics with slicing [RR15] and stream processing [RST24].

Opportunity. Although some of these approaches are flexible enough to enable the quantitative analysis of system behaviors beyond finite-state, the theoretical properties of such specifications and monitors have not been thoroughly studied before.

1.2.2 Theories of Monitorability

Safety and liveness [Lam77, AS85] are fundamental concepts in computation and serve as the foundation for many verification paradigms. The safety-liveness classification of boolean properties characterizes whether a property can be falsified by observing a finite prefix of an infinite computation trace—always for safety, never for liveness. The orthogonality of safety and liveness leads to the following celebrated fact: every property can be written as the intersection of a safety property and a liveness property [AS85, AS87]. This means that every property can be decomposed into two parts: a safety part, which is amenable to simple verification techniques such as invariants, and a liveness part, which requires heavier verification paradigms, such as ranking functions.

From a topological perspective, safety and liveness properties have natural characterizations in the Cantor topology on infinite traces: safety properties correspond to closed sets, while liveness properties are dense [AS85]. The Borel hierarchy induced by this topology—the safety-progress classification [CMP93]—reflects the increasing complexity of verification. At the first level, we find safety and co-safety properties, the latter being those whose falsehood (rather than truth)

can always be refuted after a finite number of observations. More sophisticated verification techniques are required at the second level, which consists of countable boolean combinations of first-level properties, including response and persistence properties.

The notion of monitorability bridges the gap between the semantics of a specification and its recognizability during system execution. The development of a formal theory of monitorability has been fundamentally shaped by the safety-liveness dichotomy and the safety-progress hierarchy, just as these concepts have influenced the verification [EN98, KV01] and testing [ADX01, RMT+04] of system properties.

The first formal definition of monitorability, introduced by Kim et al. [KKL+02], focused on the detection of property violations. This definition captures a strict subset of safety properties over infinite words, as it requires that the set of finite prefixes satisfying the property be co-recursively enumerable. In contrast, the broader notion of safety is not inherently constrained by computational feasibility.

The definition by Kim et al. was later generalized by Pnueli and Zaks [PZ06] considering both satisfactions and violations. According to their definition, a property is s-*monitorable* for a finite trace s if there exists a finite continuation r such that either every infinite continuation to sr satisfies the property or every infinite continuation violates it.

Building on the definition by Pnueli and Zaks, Bauer et al.[BLS11] defined a property as monitorable if it is s-monitorable for every finite trace s. Accordingly, a monitor outputs one of the following verdicts on every finite trace s: "true" if every infinite continuation to s satisfies the property, "false" if every infinite continuation to s violates the property, and "inconclusive" otherwise. This formulation is now regarded as the classical definition of monitorability. The authors demonstrated that the set of monitorable properties under this definition strictly contains the union of safety and co-safety properties. This result was further strengthened by Falcone et al.[FFM12], who showed that it also strictly contains finite boolean combinations of safety and co-safety properties. Additionally, Diekert and Leucker [DL14] provided a topological characterization of monitorable properties (in the Cantor topology of infinite words), identifying them as sets whose boundary is nowhere dense.

Extending the ideas introduced by Bauer et al.[BLS07], Falcone et al.[FFM12] proposed a generalized definition of monitorability based on parameterized truth domains rather than restricting monitor verdicts to a fixed three- or four-valued domains. This generalization enables a more refined classification of monitorable properties depending on the chosen truth domain. The authors show how their framework relates with the safety-progress hierarchy considering several truth domains. Notably, according to their definition, every (linear-time) property is monitorable in a four-valued domain where the usual "inconclusive" verdict is refined into "currently true" and "currently false" verdicts, provided that the specification formalism admits a finite-sequence semantics and that membership of finite execution sequences with respect to the property is decidable.

A recent line of work by Havelund and Peled [HP18, PH18, HP22, HP23] introduced a framework for monitorability that classifies properties based on whether they can *always, never, or sometimes* be *satisfied or refuted* by observing finite execution prefixes. For example, in this framework, safety properties are precisely those that are *always finitely refutable* and liveness properties *never finitely refutable*. Gorostiaga and Sánchez [GS22, GS24] later extended this approach to properties over richer value domains, including quantitative properties. The key difference is to characterize monitorability in terms of the "dismissibility" of potential values from a given set as longer prefixes of an execution are observed. When instantiated in the

boolean setting, for instance, safety properties are exactly those where all values strictly greater than an infinite word's actual value are *all finitely dismissible* and liveness properties *none finitely dismissible*. Expanding on this idea, the authors explored an even finer classification by considering finite dismissibility or non-dismissibility for some or all executions and for some or all values in a given set.

In addition to these definitions, Aceto et al. [AAF+19a] developed a theoretical framework of monitorability covering both linear- and branching-time settings, while Francalanza [Fra21] introduced a process-algebraic theory of monitoring.

Opportunity. Despite the various efforts to develop a theory of monitorability, almost all of these approaches focus exclusively on boolean properties, and none of them considers approximate monitoring and the inherent tradeoffs between resource use and precision as core principles.

1.2.3 Quantitative Properties and Monitoring

System properties can be quantitative in several ways. First, they may specify *absolute* measures such as the average response time of a server or the maximal memory consumption of a process. In this context, quantitative properties (a.k.a. quantitative languages) generalize classical boolean specifications by mapping execution traces—finite or infinite—to a richer domain (e.g., reals) instead of true or false. A formalism for quantitative properties is provided by quantitative automata Quantitative automata [CDH10b], a formalism for studying finite-state quantitative properties, extend finite-state automata with numerical weights on transitions and a "value function" to aggregate an infinite sequence of weights into a single real value. Common value functions include limit superior, which generalizes the Büchi acceptance condition; limit average (a.k.a. mean payoff), which computes the long-term average of weight sequences; and discounted sum, which computes the weighted sum of weight sequences where the future values contribute less due to discounting. The expressiveness, closure properties, decision problems of quantitative automata as well as their connections to game theory were initially explored in [CDH10a, CDH10b]. Later work developed and investigated extensions of quantitative automata with nesting [CHO17], non-testable counters [CHO16] and a logic thereof [Pau17], probabilistic semantics [CHO19], and alternation [CDH09]. Extensions of linear temporal logic and computation tree logic with accumulation assertions were studied in [BCHK14].

The classes of limit-average and discounted-sum automata have received particular attention. The universality problem is undecidable for nondeterministic limit-average automata [DDG+10, HPPR18] and has been long open for discounted-sum, with connections to open problems from other research fields [BHO15], in particular the reachability in piecewise-affine maps [AMP95, KPS08], membership in generalized Cantor sets [Din01, ORS16], and representations of real numbers in nonintegral bases [Rén57]. As the analysis of these models proved challenging, an exploration of variants with desirable properties was set. Mean-payoff automaton expressions [CDE+10], for example, are compositions of deterministic limit-average automata with sum, minimum, and maximum, and thus preserve their desirable properties such as robustness with respect to closure properties and decidability of universality and inclusion. Similarly, discounted-sum automata with the discount factors restricted to integers [BH14] are determinizable and their decision problems are decidable, which motivated further studies focusing on variants supporting multiple [BH21, BH23] or real-valued [Bok24] discount factors.

The idea of automata specifying functions from words to values goes back to the seminal work of Schützenberger [Sch61]: Weighted automata over finite words are finite-state models with weighted edges, where the weights are aggregated within the algebraic structure of a semiring. While this model is well understood and extensively studied [DKV09, DK21, ABK22], it still constitutes a vibrant research field. For example, a recent breakthrough showed that, for weighted automata over fields, checking whether a deterministic (resp. unambiguous) equivalent exists is decidable [BS23]. Following the introduction of quantitative automata [CDH10b], weighted automata and logics on infinite words (moving from semirings to ω-valuation monoids) has been developed in [DM12].

Cost register automata [ADD+13] are an alternative model for defining quantitative properties on finite words, which originated in the study of streaming transducers [AC10, AC11]. A cost register automaton is a finite-state automaton equipped with non-testable registers that are updated by arithmetic operations on input symbols within a cost domain. This model is expressively equivalent to weighted automata [ADD+13, AFM+20] and has corresponding characterizations as transformations definable in monadic second-order logic [AC10, AFT12, AD17, ADD+13]. Other models related to cost register automata include quantitative regular expressions [AFR16, MRA+17], an extension of regular expressions with streaming composition and numeric aggregation for specification of quantitative stream queries, and data transducers [AMS19], an automaton-based formalism with (a finite number of) data registers that is expressively equivalent to weighted automata but can be exponentially more succinct. A quantitative stream processing theory based on cost register automata was developed in [AFM+20], which focuses on runtime decidability issues for properties over data streams.

A complementary *relative* view of quantitative properties measures the distance of a given execution to a boolean property rather than aggregating numeric values along the execution. While the formalisms discussed above support absolute, aggregation-style evaluations, they also mostly accommodate this relative approach that has been a focus not only in model checking but also in runtime monitoring.

One research direction taking this relative view focused on distance-based comparisons of systems through games. The simulation distances framework [CHR12] generalizes the classical simulation preorder to a quantitative metric to compute a distance (capturing correctness, coverage, and robustness) between systems. A main motivation there is to synthesize optimal systems by minimizing such distances, which was also considered in other works both in program synthesis [BCHJ09] and repair [DSS16]. An alternative to this framework [FL14] lifts van Glabbeek's (qualitative) linear-time branching-time spectrum [vG93, vG01] into a continuum of system distances in a distance-agnostic manner.

Another line of work focuses on temporal logics with ways to measure the quality of satisfaction. For example, an extension of linear temporal logic with a quantitative semantics is studied in [FLS08], discounting in [ABK14], and averaging in [BMM14]. A relevant collection of literature on quantitative monitoring from the relative perspective features primarily the metric temporal logic [AH93] and the signal temporal logic [MN04]. For example, a quantitative semantics for metric temporal logic was introduced in [FP06, FP09] and, based on this semantics, an algebraic framework for monitoring continuous-time signals against metric temporal logic specifications in [MCW21a, MCW21b]. In [DFM13], the authors study the notion of robustness for signal temporal and provide an algorithm for its efficient computation. The key ingredient of their monitoring method is an online algorithm that computes the extremal values on a sliding window. Quantitative monitoring of signal temporal logic using edit distance between behaviors is studied in [JBG+18], which also has a flavor of approximate

monitoring due to the use of quantization of real-valued signals. In the context of monitoring real-valued signal, some tradeoffs regarding monitor quality also arise naturally due to the processing of continuous signals. For example, it is shown in [APM19] that sampling rate and the quality of robustness computation is tightly connected.

Opportunity. Even though the monitoring of properties augmented with various forms of quantities has been studied, a classification of quantitative properties and automata from a monitoring perspective is still needed.

1.2.4 Thesis Goals

In light of the opportunities we identified above, this thesis aims to achieve the following goals:

1. Develop a formal theory of quantitative monitorability enabling systematic reasoning about precision-resource tradeoffs of quantitative monitors.
2. Establish a monitoring-oriented classification of quantitative properties through quantitative extensions of safety and liveness.
3. Implement and evaluate these theoretical contributions in a general-purpose software tool supporting automated analysis and monitoring of quantitative automata.

1.3 Thesis Outline and Contributions

In Chapter 2, we formally define the necessary mathematical preliminaries. The rest of this thesis consist of three main parts. First, in Chapters 3 and 4, we develop the foundational framework for quantitative and approximate monitoring, resource use of quantitative monitors, and their resource-precision trade-offs. Next, in Chapter 5, we define and explore the notions of safety and liveness for quantitative properties and automata from the monitoring perspective. Finally, building on the theoretical contributions from previous chapters, in Chapter 6 we present the first general-purpose software tool for the analysis and monitoring of quantitative automata.

We summarize the specific contributions of each chapter below.

In Chapter 3, we introduce a theoretical framework for the "limit monitoring" of quantitative properties. In this limit monitoring setting, a monitor incrementally observes the finite prefixes of an infinite word w while producing a verdict sequence whose upper or lower limits give "estimate" for the value of w. We focus on the theoretical guarantees on the (relative) quality of these estimates in relation to the amount of resources the monitor has. Key contributions are as follows.

- We formally define several modalities of quantitative and approximate limit monitorability and show that they conservatively generalize various definitions of boolean monitorability.
- Based on these definitions, we establish a framework to evaluate monitors through precision-resource trade-offs, emphasizing the relation between approximation quality and computational resources.
- We demonstrate that additional monitor resources such as registers or states lead to strictly better approximations for certain quantitative properties.

In Chapter 4, we extend and make finer the quantitative monitoring framework from Chapter 3 in two directions. First, we propose an abstract interpretation of monitors where, for each natural number n, the aggregate semantics of a monitor at time n is an equivalence relation over all sequences of at most n observations so that two equivalent sequences are indistinguishable to the monitor and thus mapped to the same verdict value. Second, we consider in addition the "prompt monitoring" setting where a monitor is required to conform to an absolute quality measure not only on infinite words but also on their finite prefixes. Key contributions are as follows.

- We introduce an abstract interpretation of quantitative monitors for measuring monitor precision and resource use through equivalence class abstractions.
- We show that resource-optimal approximate monitors are not unique and cannot be greedily constructed from their precise counterparts.
- We demonstrate how our framework can be used for formally analyzing the resource-precision tradeoffs of quantitative monitors. For example, we show a class of properties with an infinite hierarchy of approximate monitors and identify another where it is not possible to save resources without increasing the monitor's limit error.

In the first part of Chapter 5, we extend the classical concepts of safety and liveness to the quantitative setting. These definitions build our monitoring-oriented view of quantitative properties: a property is safe when every wrong lower bound hypothesis can be falsified by observing a finite prefix of an infinite computation trace, and live when some wrong hypothesis cannot be finitely falsified. Key contributions are as follows.

- We formally define safety and liveness (as well as co-safety and co-liveness) of quantitative properties and prove that every quantitative property is the pointwise minimum of a safety property and a liveness property (similarly, the pointwise maximum of a co-safety property and a co-liveness property).
- We identify the relation between quantitative safety and topological continuity together with characterizations linking quantitative safety and liveness to their boolean analogs.
- We introduce approximate safety and co-safety and prove that for every quantitative property that is approximately safe and approximately co-safe there is an arbitrarily precise finite-state monitor.

In the second part of Chapter 5, we instantiate our quantitative safety and liveness framework with the specific classes of quantitative properties expressed by automata. Similarly to how boolean automata define classes of boolean properties amenable to boolean verification, quantitative automata define classes of quantitative properties amenable to quantitative verification. Quantitative automata generalize standard boolean automata with rational-valued transitions and a value function that accumulates an infinite sequence of weights into a single value, defining functions from infinite words into the totally-ordered domain of real numbers. Key contributions are as follows.

- We identify that checking whether a quantitative automaton defines a constant function is closely related with checking its safety and liveness and we provide algorithms solving this problem for the common classes of quantitative automata.

- We describe algorithms for constructing the safety closure of quantitative automata, checking their safety and liveness, and decomposing them into their safety and liveness components.
- We provide a comprehensive complexity analysis of these problems and the algorithms we present.

In Chapter 6, we present Quantitative Automata Kit (QuAK), the first comprehensive tool designed for automating the analysis and monitoring of quantitative automata. Key contributions are as follows.

- We implement the standard quantitative automata algorithms from the literature (e.g., for checking nonemptiness, universality, and inclusion) as well as the novel algorithms presented in Chapter 5 together with simple monitoring capabilities.
- We demonstrate QuAK's effectiveness through empirical evaluation.

Finally, in Chapter 7, we conclude the thesis with a discussion on future research directions and potential impact.

CHAPTER 2

Definitions

2.1 Basic sets

We respectively denote by $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, and $\mathbb{R}$ the set of nonnegative integers (i.e., naturals), integers, rationals, and reals. In addition, we define the extended sets $\overline{\mathbb{N}} = \mathbb{N} \cup \{+\infty\}$, $\overline{\mathbb{Z}} = \mathbb{Z} \cup \{-\infty, +\infty\}$, and $\overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty, +\infty\}$. When it is clear from the context, we simply write ∞ instead of $+\infty$.

2.2 Order-theoretic structures

A *complete lattice* is a partially-ordered set where all subsets have a supremum (a.k.a. least upper bound or join) and an infimum (a.k.a. greatest lower bound or meet). Given a complete lattice $\mathbb{D}$, we write $\bot$ for $\inf \mathbb{D}$ and $\top$ for $\sup \mathbb{D}$. Unless otherwise stated, we assume throughout the thesis that complete lattices are nontrivial, i.e., $\bot \neq \top$. Whenever appropriate, we write 0 or $-\infty$ instead of $\bot$, and 1 or ∞ instead of $\top$. We respectively use the terms minimum and maximum (resp. infimum and supremum) for the greatest lower bound and the least upper bound of finitely (resp. infinitely) many elements. The *inverse* (a.k.a. opposite or dual) of $\mathbb{D}$ is the complete lattice $\mathbb{D}_{inv}$ that contains the same elements as $\mathbb{D}$ but with the ordering reversed.

2.3 Alphabets and words

Let $\Sigma = \{a, b, \ldots\}$ be a finite alphabet of letters (observations). A *word* (a.k.a. *trace*) over Σ is a sequence of letters from Σ. We denote by Σ^* the set of *finite* words and by Σ^ω the set of *infinite* words. For $n \in \mathbb{N}$, we denote by Σ^n the set of finite words of length n and by $\Sigma^{\leq n}$ those of length at most n. Given $u \in \Sigma^*$ and $w \in \Sigma^* \cup \Sigma^\omega$, we write $u \prec w$ (resp. $u \preceq w$) when u is a strict (resp. nonstrict) prefix of w. We denote by $|w|$ the length of $w \in \Sigma^* \cup \Sigma^\omega$ and, given $a \in \Sigma$, by $|w|_a$ the number of occurrences of a in w. For $w \in \Sigma^* \cup \Sigma^\omega$ and $0 \leq i < |w|$, we denote by $w[i]$ the ith letter of w.

2.4 Boolean and quantitative properties

A *boolean property* $P \subseteq \Sigma^\omega$ is a set of infinite words. A *value domain* $\mathbb{D}$ is a partially-ordered set, which we assume to be a complete lattice unless otherwise stated. A *quantitative property* is a total function $\Phi : \Sigma^\omega \to \mathbb{D}$ from the set of infinite words to a value domain. We use the boolean domain $\mathbb{B} = \{0, 1\}$ with $0 < 1$ and, in place of P, its characteristic property $\Phi_P : \Sigma^\omega \to \mathbb{B}$, which is defined by $\Phi_P(w) = 1$ if $w \in P$, and $\Phi_P(w) = 0$ if $w \notin P$. When we say just *property*, we mean a quantitative one. Let $n \geq 1$ be an integer and $\Phi_1, \ldots, \Phi_n$ be properties on $\mathbb{D}$. Given a function $f : \mathbb{D}^n \to \mathbb{D}$, we define $f(\Phi_1, \ldots, \Phi_n)(w) = f(\Phi_1(w), \ldots, \Phi_n(w))$ all infinite words $w \in \Sigma^\omega$.

Recall that the inverse of a value domain $\mathbb{D}$ is denoted by $\mathbb{D}_{inv}$ and it contains the same elements as $\mathbb{D}$ but with the ordering reversed. For a property $\Phi : \Sigma^\omega \to \mathbb{D}$, its *complement* $\overline{\Phi}$ maps every infinite word to the same value in the inverse of $\mathbb{D}$, i.e., $\overline{\Phi} : \Sigma^\omega \to \mathbb{D}_{inv}$ is defined as $\overline{\Phi}(w) = \Phi(w)$ for all $w \in \Sigma^\omega$. Given a property $\Phi : \Sigma^\omega \to \mathbb{D}$ and a value $v \in \mathbb{D}$, we define $\Phi_{\sim v} = \{w \in \Sigma^\omega \mid \Phi(w) \sim v\}$ for $\sim \in \{\leq, \geq, \not\leq, \not\geq, <, >, =\}$. The *top value* of a property Φ is $\sup_{w \in \Sigma^\omega} \Phi(w)$, which we denote by $\top_\Phi$, and its *bottom value* is $\bot_\Phi = \inf_{w \in \Sigma^\omega} \Phi(w)$. For all properties Φ_1, Φ_2 on a value domain $\mathbb{D}$ and all words $w \in \Sigma^\omega$, we let $\min(\Phi_1, \Phi_2)(w) = \min(\Phi_1(w), \Phi_2(w))$ and $\max(\Phi_1, \Phi_2)(w) = \max(\Phi_1(w), \Phi_2(w))$.

Some properties can be defined as limits of value sequences. A *finitary property* $\pi \colon \Sigma^* \to \mathbb{D}$ associates a value with each finite word. A *value function* $\mathsf{Val} \colon \mathbb{D}^\omega \to \mathbb{D}$ accumulates an infinite sequence of values to a single value. Given a finitary property π, a value function Val, and a word $w \in \Sigma^\omega$, we write $\mathsf{Val}_{u \prec w} \pi(u)$ instead of $\mathsf{Val}(\pi(u_0)\pi(u_1)\ldots)$, where each u_i satisfies $u_i \prec w$ and $|u_i| = i$.

A property $\Phi : \Sigma^\omega \to \mathbb{D}$ is a *limit property* when there exists a finitary property $\pi : \Sigma^* \to \mathbb{D}$ and a value function $\mathsf{Val} : \mathbb{D}^\omega \to \mathbb{D}$ such that $\Phi(w) = \mathsf{Val}_{u \prec w} \pi(u)$ for all $w \in \Sigma^\omega$. We denote this by $\Phi = (\pi, \mathsf{Val})$. In particular, if $\Phi = (\pi, \mathsf{Val})$ for $\mathsf{Val} \in \{\mathsf{Inf}, \mathsf{Sup}, \mathsf{LimInf}, \mathsf{LimSup}\}$, then Φ is a Val-*property*.

2.5 Quantitative Automata

A *nondeterministic quantitative automaton* [CDH10b] (or just automaton from here on) on words is a tuple $\mathcal{A} = (\Sigma, Q, \iota, \delta)$, where Σ is an alphabet; Q is a finite nonempty set of states; $\iota \in Q$ is an initial state; and $\delta \colon Q \times \Sigma \to 2^{(\mathbb{Q} \times Q)}$ is a finite transition function over weight-state pairs. A *transition* is a tuple $(q, \sigma, x, q') \in Q \times \Sigma \times \mathbb{Q} \times Q$, such that $(x, q') \in \delta(q, \sigma)$, also written $q \xrightarrow{\sigma:x} q'$. (There might be finitely many transitions with different weights over the same letter between the same states.[1]) We write $\gamma(t) = x$ for the weight of a transition $t = (q, \sigma, x, q')$. $\mathcal{A}$ is deterministic if for all $q \in Q$ and $\sigma \in \Sigma$, the set $\delta(q, \sigma)$ is a singleton. We require the automaton $\mathcal{A}$ to be *total* (a.k.a. complete), namely that for every state $q \in Q$ and letter $\sigma \in \Sigma$, there is at least one state q' and a transition $q \xrightarrow{\sigma:x} q'$. For a state $q \in Q$, we denote by $\mathcal{A}^q$ the automaton that is derived from $\mathcal{A}$ by setting its initial state ι to q.

A run of $\mathcal{A}$ on a word w is a sequence $\rho = q_0 \xrightarrow{w[0]:x_0} q_1 \xrightarrow{w[1]:x_1} q_2 \ldots$ of transitions where $q_0 = \iota$ and $(x_i, q_{i+1}) \in \delta(q_i, w[i])$. For $0 \leq i < |w|$, we denote the ith transition in ρ by $\rho[i]$, and the finite prefix of ρ up to and including the ith transition by $\rho[..i]$. As each transition t_i carries a weight $\gamma(t_i) \in \mathbb{Q}$, the sequence ρ provides a weight sequence $\gamma(\rho) = \gamma(t_0)\gamma(t_1)\ldots$

[1] The flexibility of allowing "parallel" transitions with different weights is often omitted, as it is redundant for some value functions, including the ones we focus on in the sequel, while important for others.

A Val-automaton is one equipped with a value function Val : $\mathbb{Q}^\omega \to \mathbb{R}$, which assigns real values to runs of $\mathcal{A}$. We assume that Val is bounded for every finite set of rationals, i.e., for every finite $V \subset \mathbb{Q}$ there exist $m, M \in \mathbb{R}$ such that $m \leq \mathsf{Val}(x) \leq M$ for every $x \in V^\omega$. The finite set V corresponds to transition weights of a quantitative automaton, and the concrete value functions we consider satisfy this assumption (see below).

Notice that while quantitative properties can be defined over arbitrary value domains, we restrict quantitative automata to totally-ordered numerical value domains (i.e., bounded subsets of $\mathbb{R}$) as this is the standard setting in the literature.

The value of a run ρ is $\mathsf{Val}(\gamma(\rho))$. The value of a Val-automaton $\mathcal{A}$ on a word w, denoted $\mathcal{A}(w)$, is the supremum of $\mathsf{Val}(\rho)$ over all runs ρ of $\mathcal{A}$ on w, generalizing the standard approach in boolean automata where acceptance is defined through the existence of an accepting run. The *top value* of a Val-automaton $\mathcal{A}$ is $\top_\mathcal{A} = \sup_{w \in \Sigma^\omega} \mathcal{A}(w)$, and its *bottom value* is $\bot_\mathcal{A} = \inf_{w \in \Sigma^\omega} \mathcal{A}(w)$, which we denote by $\top$ and $\bot$ when $\mathcal{A}$ is clear from the context. Note that when we speak of the top value of an automaton or a property expressed by an automaton, we always match its value domain to have the same top value. The size of an automaton consists of the maximum among the size of its alphabet, state space, and transition space, where weights are represented in binary.

We list below the value functions for quantitative automata that we will use, defined over infinite sequences $x = x_0 x_1 \ldots$ of rational weights.

- $\mathsf{Inf}(x) = \inf\{x_n \mid n \geq 0\}$
- $\mathsf{Sup}(x) = \sup\{x_n \mid n \geq 0\}$
- $\mathsf{LimInf}(x) = \lim_{n\to\infty} \inf\{x_i \mid i \geq n\}$
- $\mathsf{LimSup}(x) = \lim_{n\to\infty} \sup\{x_i \mid i \geq n\}$
- $\mathsf{LimInfAvg}(x) = \mathsf{LimInf}\left(\frac{1}{n}\sum_{i=0}^{n-1} x_i\right)$
- $\mathsf{LimSupAvg}(x) = \mathsf{LimSup}\left(\frac{1}{n}\sum_{i=0}^{n-1} x_i\right)$
- For a discount factor $\lambda \in \mathbb{Q} \cap (0,1)$, $\mathsf{DSum}_\lambda(x) = \sum_{i\geq 0} \lambda^i x_i$

Note that (i) when the discount factor $\lambda \in \mathbb{Q} \cap (0,1)$ is unspecified, we write DSum, and (ii) LimInfAvg and LimSupAvg are also called $\underline{\text{MeanPayoff}}$ and $\overline{\text{MeanPayoff}}$ in the literature.

The basic decision problems for boolean automata extend to this model naturally. Consider two quantitative automata $\mathcal{A}$ and $\mathcal{B}$ together with a rational threshold $v \in \mathbb{Q}$.

- $\mathcal{A}$ is *nonempty with respect to* v iff $\mathcal{A}(w) \geq v$ for some $w \in \Sigma^\omega$.
- $\mathcal{A}$ is *universal with respect to* v iff $\mathcal{A}(w) \geq v$ for all $w \in \Sigma^\omega$.
- $\mathcal{A}$ is *included in* $\mathcal{B}$ iff $\mathcal{A}(w) \leq \mathcal{B}(w)$ for all $w \in \Sigma^\omega$.
- $\mathcal{A}$ and $\mathcal{B}$ are *equivalent* iff $\mathcal{A}(w) = \mathcal{B}(w)$ for all $w \in \Sigma^\omega$.

In this thesis, we use the term *quantitative automata* rather than *weighted automata*, following the distinction made in [Bok21]: Quantitative automata retain the classical "preference" interpretation of nondeterministic branching: existential choices are evaluated by the supremum and universal choices by the infimum, which forces both the weight domain and its associated value functions to form a complete lattice (e.g., a bounded subset of $\mathbb{R}$). By contrast,

weighted automata abstract away this branching semantics in favor of an arbitrary commutative aggregation operator supplied by a semiring or valuation monoid, generally without any dual (universality) interpretation. Consequently, weighted automata allow weights and value functions from arbitrary domains.

CHAPTER 3

Quantitative and Approximate Limit Monitoring

In this chapter, the following publications were re-used in full:

- Thomas A. Henzinger, N. Ege Saraç. *Quantitative and Approximate Monitoring*. In 36th Annual ACM/IEEE Symposium on Logic in Computer Science, **LICS 2021**.

3.1 Introduction

We provide a theoretical framework for the convergence of two recent trends in computer-aided verification. The first trend is *runtime verification* [BFFR18]. Classical verification aspires to provide a judgment about all possible runs of a system; runtime verification, or *monitoring*, provides a judgment about a single, given run. There is a trend towards monitoring because the classical "verification gap" keeps widening: while verification capabilities are increasing, system complexity is increasing more quickly, especially in the time of many-core processors, cloud computing, cyber-physical systems, and neural networks. Theoretically speaking, the paradigmatic classical verification problem is *emptiness* of the product between system and negated specification ("does some run of the given system violate the given specification?"), whereas the central runtime verification problem is *membership* ("does a given run satisfy a given specification?"). Since membership is easier to solve than emptiness, this has ramifications for specification formalisms; in particular, there is no need to restrict ourselves to ω-regular specifications or finite-state monitors. We do restrict ourselves to the *online* setting, where a monitor watches the finite prefixes of an infinite run and, with each prefix, renders a *verdict*, which could signal a satisfaction or violation of the specification, or "don't know yet."

The second trend is *quantitative verification* [Kwi07, Hen13]. While classical verification is boolean, in that every complete run either satisfies or violates the specification and, accordingly, the system is either correct (i.e., without a violating run) or incorrect, quantitative verification provides additional, often numerical information about runs and systems. For example, a quantitative specification may measure the probability of an event, the "response time" or the use of some other resource along a run, or by how much a run deviates from a correct run. In *quantitative runtime verification*, we wish to observe, for instance, the maximal or

average response time along a given run, not across all possible runs. Quantitative verification is interesting for an important reason beyond its ability to provide non-boolean information: it may provide *approximate* results [BH14]. A monitor that under- or over-approximates a quantitative property may be able to do so with fewer computational resources than a monitor that computes a quantitative property's exact value. We provide a theoretical framework for *quantitative and approximate monitoring*, which allows us to formulate and prove such statements.

In boolean runtime verification frameworks, there are several different notions of *monitorability* [KKL+02, FFM12, AAF+19b]. Along with safety and co-safety, a well-studied definition is by [PZ06] and [BLS11]: after watching any finite prefix of a run, if a positive or negative verdict has not been reached already, there exists at least one continuation of the run which will allow such a verdict. This *existential* definition is popular because a *universal* definition, that on every run a positive or negative verdict will be reached eventually, is very restrictive; only boolean properties that are both safe and co-safe can be monitored universally [AAF+19b]. By contrast, the existential definition covers finite boolean combinations of safety and co-safety, and more [FFM12]. In *quantitative approximate* monitoring, however, there is less need to prefer an existential definition of monitoring because usually many approximations are available, even if some are poor. The main attention must shift, rather, to the quality—i.e., precision—of the approximation. Our quantitative framework fully generalizes the standard boolean versions of monitorability in a universal setting where monitors yield approximate results on all runs and can be compared regarding their precision and resource use. In fact, we advocate the consideration of precision-resource tradeoffs as a central design criterion for monitors, which requires a formalization of monitoring in which precision-resource tradeoffs can be analyzed. Such a formalization is the main contribution of this chapter.

As an example, let us illustrate a precision-resource tradeoff that occurs when using *register machines* as monitors. Consider a server that processes requests. Each trace of the server is an infinite word over the alphabet $\{\mathtt{rq}, \mathtt{ack}, \mathtt{oo}\}$ of events. An interesting quantitative property of the server is *maximal response time*, which measures the maximal number of events before each rq event in a trace is followed by an ack event. This property, denoted Φ_1, is a function that maps every infinite word to a value in $\mathbb{N} \cup \{\infty\}$. To construct a precise online monitor for Φ_1, we need two counter registers x and y and the ability to compare their values: as long as $x < y$, register x counts the current response time, and y stores the maximal response time encountered so far; if $x = y$, counting continues in y, and x is reset to 0. The output, or *verdict* value, of the monitor is always y. In this way the 2-counter monitor M_{max} generates the verdict function depicted in Figure 3.1.

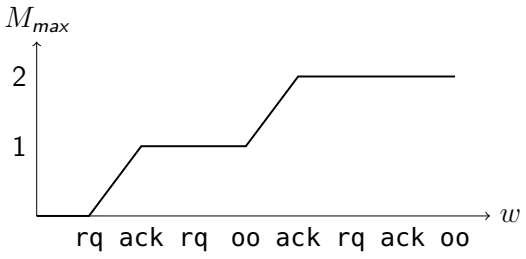


Figure 3.1: Monitoring maximal response time for a trace w.

Considering the same server, one may also be interested in the *average response time* of a trace. The precise monitoring of average response time requires 3 counters and division between counter registers to generate outputs. Moreover, verdict values can fluctuate along

a trace, producing a non-monotonic verdict function. Figure 3.2 shows the verdict function generated by a 3-register monitor M_{avg} with division.

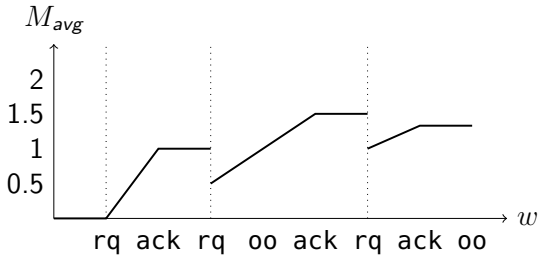


Figure 3.2: Monitoring average response time for a trace w.

Now, let us consider an alphabet $\{\mathsf{rq}_1, \mathsf{ack}_1, \mathsf{rq}_2, \mathsf{ack}_2, \mathsf{oo}\}$ with two types of matching $(\mathsf{rq}_i, \mathsf{ack}_i)$ pairs. The quantitative property Φ_2 measures the maximal response times for both pairs: it maps every trace to an ordered pair of values from $\mathbb{N} \cup \{\infty\}$. A construction similar to the one for Φ_1 gives us a precise monitor that uses 4 counters. Indeed, we will show that 3 counters do not suffice to monitor Φ_2 precisely. However, the quantitative property Φ_2 can be approximately monitored with 3 counters: two counters can be used to store the maxima so far, and the third counter may track the current response time prioritizing the pair $(\mathsf{rq}_1, \mathsf{ack}_1)$ whenever both request types are active. This 3-counter monitor will always under-approximate the maximal response time for the $(\mathsf{rq}_2, \mathsf{ack}_2)$ pair. In case the resources are even scarcer, a 2-counter monitor can keep the same value as an under-approximation for both maximal response times in one counter, and use the second counter to wait sequentially for witnessing $(\mathsf{rq}, \mathsf{ack})$ pairs of both types before incrementing the first counter. Just like the number of registers leads to precision-resource tradeoffs for register monitors, the number of states leads to precision-resource tradeoffs for finite-state monitors. For instance, a fixed number of states can encode counter values up to a certain magnitude, but can under-approximate larger values. We provide a general formal framework for quantitative and approximate monitoring which allows us to study such tradeoffs for different models of monitors.

In Section 3.2, we define quantitative properties, approximate verdict functions, and how the precision of monitors can be compared. In Section 3.3, we give a variety of different examples and closure operations for quantitative monitoring. We also characterize the power of the important class of *monotonic* monitors by showing that, in our framework, the quantitative properties that can be monitored universally (on all traces) and precisely by monotonically increasing verdict functions are exactly properties that are "continuous from below" on the value domain. In Section 3.4, we embed several variations of the boolean value domain within our quantitative framework. This allows us to characterize, within the safety-progress hierarchy [CMP93], which boolean properties can be monitored universally and existentially; see Tables 3.1 and 3.2. The section also connects our quantitative definitions of monitorability to the boolean definitions of [FFM12, AAF+19b, PZ06, BLS11] and shows that our quantitative framework generalizes their popular boolean settings conservatively. Finally, in Section 3.5, we present precision-resource tradeoffs for register monitors. For this purpose, we generalize the quantitative setting of [FHS18, FHK20] to approximate monitoring within our framework. In particular, we show a family of quantitative properties for which every additional counter register improves the monitoring precision.

Related work. In the boolean setting, the first definition of *monitorability* [KKL+02] focused on detecting violations of a property. This definition was generalized by [PZ06] and [BLS11] to

capture satisfactions as well. Later, instead of using a fixed, three-valued domain for monitoring, Falcone et al. [FFM12] proposed a definition with parameterized truth domains. According to their definition, every linear-time property is monitorable in a four-valued domain where the usual "inconclusive" verdict is split into "currently true" and "currently false" verdicts. Frameworks that capture existential as well as universal modalities for monitorability were studied in a branching-time setting [FAA+17, AAF+19a].

The prevalence of LTL and ω-regular specifications in formal verification is also reflected in runtime verification [BLS11, BLS06, BLS07]. Recently, several more expressive models have been proposed, such as register monitors [FHS18], monitors for visibly pushdown languages [DLT13a], quantified event automata [BFH+12], and many others for monitoring data events over an infinite alphabet of observations, as surveyed in [HRTZ18]. One step towards quantitative properties is the augmenting of boolean specifications with quantities, e.g., discounting and averaging modalities [dAHM03, BMM14], timed specifications [BNF13, PJT+17], or specifications that include continuous signals, particularly in the context of cyber-physical systems [BFMU17, JBG+18]. Another prominent line of work that provides a framework for runtime verification beyond finite-state is that of Alur et al. [AMS17, AMS19, AFM+20]. Their work focuses on runtime decidability issues for boolean specifications over streams of data events, but they do not consider approximate monitoring at varying degrees of precision. Quantitative frameworks for comparing traces and implementations for the same boolean specification were studied in [CB02, BCHJ09]. Our approach is fundamentally different as we consider quantitative property values.

Although quantitative aspects of systems have been studied much in the context of probabilistic model checking [Kwi07], decision problems in verification for quantitative properties (a.k.a. quantitative languages) [CDH10b], and games with quantitative objectives [BCJ18, BMR+18], in runtime verification, we observe a gap. While some formalisms for monitoring certain quantitative properties have been proposed [FHK20, CHO16, Pau17], to the best of our knowledge, our work is the first general semantic framework that explores what it means to monitor and approximate generic quantitative properties of traces. We believe that such a framework is needed for the systematic study of precision-resource tradeoffs in runtime verification. See [CGKM12] for a discussion of why quantitative verification at runtime is needed for self-adapting systems, and [FP18, HLS20] for monitoring neural networks.

3.2 Definitional Framework

3.2.1 Quantitative Properties and Verdict Functions

Recall that a *value domain* $\mathbb{D}$ is a complete lattice, unless stated otherwise, and a *quantitative property* $\Phi : \Sigma^\omega \to \mathbb{D}$ is a total function from infinite traces to values. A *verdict* $\nu : \Sigma^* \to \mathbb{D}$ is a function on finite traces such that for all infinite traces $w \in \Sigma^\omega$, the set $\{\nu(u) : u \in pref(w)\}$ of verdict values over all prefixes of w has a supremum (least upper bound) and an infimum (greatest lower bound). If $\mathbb{D}$ is a complete lattice, then these limits always exist. For an infinite trace $w \in \Sigma^\omega$, we write $\nu(w) = (\nu(u_i))_{i\in\mathbb{N}}$ for the infinite *verdict sequence* over the prefixes $u_i \prec w$ of increasing length i. We use the $\limsup$ or $\liminf$ of a verdict sequence $\nu(w)$ to represent the "estimate" that the verdict function ν provides for a quantitative property value $\Phi(w)$ on the infinite trace w.

Definition 3.2.1. *Let Φ be a quantitative property and $w \in \Sigma^\omega$ an infinite trace. A verdict function ν* approximates *Φ on w from below (resp.* above*) iff $\limsup \nu(w) \leq \Phi(w)$ (resp.*

$\Phi(w) \leq \liminf \nu(w)$*). Moreover,* ν monitors Φ on w from below *(resp.* above*) iff the equality holds.*

3.2.2 Universal, Existential, and Approximate Monitorability

We define three modalities of quantitative monitorability.

Definition 3.2.2. *A quantitative property* Φ *is* universally monitorable from below *(resp.* above*) iff there exists a verdict function* ν *such that for every* $w \in \Sigma^\omega$ *we have that* ν *monitors* Φ *on* w *from below (resp. above).*

Definition 3.2.3. *A quantitative property* Φ *is* existentially monitorable from below *(resp.* above*) iff there exists a verdict function* ν *such that (i) for every* $w \in \Sigma^\omega$ *we have that* ν *approximates* Φ *on* w *from below (resp. above), and (ii) for every* $u \in \Sigma^*$ *there exists* $w \in \Sigma^\omega$ *such that* ν *monitors* Φ *on* uw *from below (resp. above).*

Definition 3.2.4. *A quantitative property* Φ *is* approximately monitorable from below *(resp.* above*) iff there exists a verdict function* ν *such that for every* $w \in \Sigma^\omega$ *we have that* ν *approximates* Φ *on* w *from below (resp. above).*

Observe that every property is trivially approximately monitorable from below or above. We demonstrate the definitions in the example below.

Example 3.2.5. *Let* $\Sigma = \{\mathsf{rq}_1, \mathsf{ack}_1, \mathsf{rq}_2, \mathsf{ack}_2, \mathsf{oo}\}$ *and* $\mathbb{D} = \overline{\mathbb{N}}$*. Consider the maximal response-time properties* Φ_1 *and* Φ_2 *over* $(\mathsf{rq}_1, \mathsf{ack}_1)$ *and* $(\mathsf{rq}_2, \mathsf{ack}_2)$ *pairs, respectively. For every* $w \in \Sigma^\omega$*, let* $\Phi(w) = \max(\Phi_1(w), \Phi_2(w))$*. Consider the verdict* ν_1 *that counts both response times and outputs the maximum of the two, the verdict* ν_2 *that counts and computes the maximum only for the* $(\mathsf{rq}_1, \mathsf{ack}_1)$ *pair, and the constant verdict* ν_3 *that always outputs 0. Evidently,* ν_1 *universally monitors* Φ *from below, and* ν_3 *approximately monitors* Φ *from below. Moreover,* ν_2 *existentially monitors* Φ *from below because the true maximum can only be greater, and we can extend every finite trace* $u \in \Sigma^*$ *with* $w = \mathsf{rq}_1 \cdot \mathsf{oo}^\omega$ *such that* $\limsup \nu_2(uw) = \Phi(uw) = \infty$*.*

3.2.3 Monotonic Verdict Functions

Of particular interest are *monotonic* verdict functions, because the "estimates" they provide for a quantitative property value are always conservative (below or above) and can improve in quality over time. On the other hand, some properties, such as average response time, inherently require non-monotonic verdict functions for universal monitoring.

Definition 3.2.6. *A verdict function* ν *is* monotonically increasing *(resp.* decreasing*) iff for every* $u, t \in \Sigma^*$ *we have* $u \prec t$ *implies* $\nu(u) \leq \nu(t)$ *(resp.* $\nu(u) \geq \nu(t)$*). Moreover,* ν *is* monotonic *iff it is either monotonically increasing or monotonically decreasing. If* ν *is monotonic or non-monotonic, then it is* unrestricted*.*

If the value domain $\mathbb{D}$ has a least and a greatest element, every monotonic verdict ν that universally monitors a property Φ from below also universally monitors Φ from above. Therefore, in such cases, we say that ν *universally monitors* Φ. In Example 3.2.5 above, the verdict ν_1 is monotonically increasing and thus universally monitors Φ. Let ν_4 be such that $\nu_4(u) = \infty$ if u contains a request that is not acknowledged, and $\nu_4(u) = \nu_1(u)$ otherwise. The verdict ν_4 is not monotonic, but it universally monitors Φ from above.

3.2.4 Comparison of Verdict Functions

Quantitative monitoring provides a natural notion of precision for verdict functions.

Definition 3.2.7. *Let Φ be a quantitative property that is (universally, existentially, or approximately) monitorable from below (resp. above) by the verdict functions ν_1 and ν_2. The verdict ν_1 is* more precise *than the verdict ν_2 iff for every $w \in \Sigma^\omega$ we have $\limsup \nu_2(w) \leq \limsup \nu_1(w)$ (resp. $\liminf \nu_1(w) \leq \liminf \nu_2(w)$) and there exists $w' \in \Sigma^\omega$ such that $\limsup \nu_2(w') < \limsup \nu_1(w')$ (resp. $\liminf \nu_1(w') < \liminf \nu_2(w')$). Moreover, ν_1 and ν_2 are* equally precise *iff for every $w \in \Sigma^\omega$ we have $\limsup \nu_2(w) = \limsup \nu_1(w)$ (resp. $\liminf \nu_1(w) = \liminf \nu_2(w)$).*

Note that for a quantitative property Φ, if the verdict functions ν_1 and ν_2 universally monitor Φ both from below or from above, then ν_1 and ν_2 are equally precise. Two monotonically increasing or monotonically decreasing verdict functions can be compared not only according to their precision but also according to their *speed*, that is, how quickly they approach the property value. This will be important if monitors have limited resources and their outputs are delayed, i.e., they affect not the current but a future verdict value.

3.3 Monitorable Quantitative Properties

3.3.1 Examples

We provide several examples of quantitative properties and investigate their monitorability.

Example 3.3.1 (Maximal response time)**.** *Let $\Sigma = \{\mathsf{rq}, \mathsf{ack}, \mathsf{oo}\}$ and $\mathbb{D} = \mathbb{N} \cup \{\infty\}$. Let* mrt $: \Sigma^* \to \mathbb{D}$ *be such that* mrt$(u) = \infty$ *if, in u, a rq is followed by another rq without an ack in between, it equals the maximal number m_u of observations between matching $(\mathsf{rq}, \mathsf{ack})$ pairs if there is no pending request in u, and otherwise it equals $\max(m_u, n)$ where n is the current response time. For every $w \in \Sigma^\omega$, let us denote by* mrt(w) *the infinite sequence* $(\mathsf{mrt}(u_i))_{i \in \mathbb{N}}$ *over the prefixes $u_i \prec w$ of increasing length i. Consider the property $\Phi(w) = \lim \mathsf{mrt}(w)$ that specifies the maximal response time of a server that can process at most one request at a time. To monitor Φ, we use* mrt *as the verdict, i.e., we let $\nu(u) =$* mrt(u) *for every $u \in \Sigma^*$. Observe that* mrt *is monotonically increasing, and the construction yields $\lim \nu(w) = \Phi(w)$ for every $w \in \Sigma^\omega$. Therefore, the verdict ν universally monitors Φ.*

The maximal response-time property of Example 3.3.1 is evidently infinite-state because it requires counting up to an arbitrarily large integer. However, there are finite-state approximations that improve in precision with every additional state. We say that a finite-state machine *generates* a verdict function iff, on every finite trace, the machine's output equals the verdict value, where an output is a mapping from the set of states to the value domain.

Example 3.3.2 (Approximate monitoring of maximal response time)**.** *Consider the maximal response-time property Φ from Example 3.3.1. Let M_k be a finite-state machine with k states, and let ν_k be the verdict generated by M_k. For every $k \in \mathbb{N}$, the best the verdict ν_k can do is to approximately monitor Φ from below, because it can only count up to some integer $m \leq k$. Suppose that we are given $k+1$ states. We can use the additional state to construct a machine M_{k+1} from M_k to generate a more precise verdict ν_{k+1} as follows. We add the appropriate transitions from the states that have the output value of m to the new state, which is assigned the output $m+1$. With the additional transitions, the machine M_{k+1} can*

continue counting for one more step after reading a trace in which the current maximum is m. Therefore, ν_{k+1} is more precise than ν_k.

Next, we define the average response-time property and present two verdict functions that illustrate another kind of precision-resource tradeoff for monitors.

Example 3.3.3 (Average response time)**.** *Let $\Sigma = \{\mathsf{rq}, \mathsf{ack}, \mathsf{oo}\}$ and $\mathbb{D} = \mathbb{R} \cup \{\infty\}$. Let* art $: \Sigma^* \to \mathbb{D}$ *be such that* $\mathsf{art}(u) = \infty$ *if u contains a* rq *followed by another* rq *without an* ack *in between, it equals the average number of observations between matching $(\mathsf{rq}, \mathsf{ack})$ pairs if there is no pending* rq *in u, and otherwise it equals $\frac{n \cdot x_n + m}{n+1}$, where n is the number of acknowledged requests, x_n is the average response time for the first n requests, and m is the number of observations since the last* rq*. For every $w \in \Sigma^\omega$, let* $\mathsf{art}(w) = (\mathsf{art}(u_i))_{i \in \mathbb{N}}$ *over the prefixes $u_i \prec w$ of increasing length i. Now, define* $\liminf \mathsf{avg}(w) = \liminf \mathsf{art}(w)$ *for every $w \in \Sigma^\omega$ [CDH10b], and let Φ be the quantitative property such that $\Phi(w) = \liminf \mathsf{avg}(w)$. In other words, Φ specifies the average response time of a server that can process at most one request at a time. To monitor Φ, we can use the function* art *as a verdict, i.e., let ν be such that $\nu(u) = \mathsf{art}(u)$ for all $u \in \Sigma^*$. Intuitively, the moving average approaches to the property value as ν observes longer prefixes. Therefore, by construction, for every $w \in \Sigma^\omega$, we have $\liminf \nu(w) = \Phi(w)$, which means that Φ is universally monitorable from above by an unrestricted verdict function.*

Alternatively, we can use the monotonic verdict function ν' that universally monitors the maximal response-time property in Example 3.3.1. Observe that ν' existentially monitors Φ from above because (i) the maximal response time of a trace is greater than its average response time, and (ii) for every finite prefix u there is an extension w that contains a request that is not acknowledged, which yields $\liminf \nu'(uw) = \Phi(uw) = \infty$.

Boolean safety and co-safety properties can be embedded in a quantitative setting by considering their *discounted* versions [dAHM03]. We show that discounted safety and co-safety properties are universally monitorable.

Example 3.3.4 (Discounted safety and co-safety)**.** *Let Φ be a discounted safety property, that is, $\Phi(w) = 1$ if w does not violate the given safety property, and $\Phi(w) = 1 - \frac{1}{2^n}$ if the shortest violating prefix of w has length n. Similarly, let Ψ be a discounted co-safety property: $\Psi(w) = 0$ if w does not satisfy the given co-safety property, and $\Psi(w) = \frac{1}{2^n}$ if the shortest satisfying prefix of w has length n. To monitor these two properties, we use verdict functions ν_Φ and ν_Ψ that work similarly as Φ and Ψ on finite traces, that is, $\nu_\Phi(u) = 1$ if u is not violating for the given safety property, and $\nu_\Phi(u) = 1 - \frac{1}{2^n}$ if the shortest violating prefix of u has length n; and similarly for ν_Ψ. One can easily verify that Φ and Ψ are universally monitorable by ν_Φ and ν_Ψ, respectively.*

Finally, we look at another classical example of quantitative properties, often called *energy* properties [CDH10b].

Example 3.3.5 (Energy)**.** *Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, \gamma)$ be a deterministic finite automaton with weighted transitions, where Q is a set of states, Σ is an alphabet, $\delta \subseteq Q \times \Sigma \times Q$ is a set of transitions, q_0 is the initial state, and $\gamma : \delta \to \mathbb{Z}$ is a weight function. Let $u = \sigma_1 \dots \sigma_n$ be a finite trace of length n, and let $q_0 \dots q_n$ be the corresponding run of $\mathcal{A}$. We define $\mathcal{A}(u) = \sum_{i=1}^{n} w(q_{i-1}, \sigma_i, q_i)$, where $A(\varepsilon) = 0$. Consider the value domain $\mathbb{D} = \mathbb{Z} \cup \{\infty\}$. Let Φ be a property such that, for every $w \in \Sigma^\omega$, we have $\Phi(w) = k$ where k is the smallest*

nonnegative value that satisfies $\mathcal{A}(u) + k \geq 0$ for every finite prefix $u \prec w$. To monitor Φ, we construct the following verdict function: given $u \in \Sigma^*$, let $\nu(u) = -\min\{\mathcal{A}(t) \mid t \in pref(u)\}$. Note that ν is monotonically increasing. On an infinite trace $w \in \Sigma^\omega$, if $\nu(w)$ approaches ∞, then w yields a negative-weight loop on $\mathcal{A}$, therefore $\Phi(w) = \infty$. Otherwise, if $\nu(w)$ converges to a finite value, then it is equal to $\Phi(w)$ by construction, which means that ν universally monitors Φ.

3.3.2 Closure under Operations on the Value Domain

Let $\mathbb{D}$ be a value domain and $\Phi : \Sigma^\omega \to \mathbb{D}$ be a quantitative property. Recall that the *complement* of Φ is a quantitative property $\overline{\Phi} : \Sigma^\omega \to \mathbb{D}_{inv}$ where $\mathbb{D}$ and $\mathbb{D}_{inv}$ contain the same elements but their ordering is reversed.

Proposition 3.3.6. *A quantitative property Φ is universally (resp. existentially; approximately) monitorable from below iff $\overline{\Phi}$ is universally (resp. existentially; approximately) monitorable from above.*

If the value domain $\mathbb{D}$ is a lattice, then monitorability from below is preserved by the least upper bound (written $\max$) and from above by greatest lower bound (written $\min$). For all quantitative properties Φ and Ψ on $\mathbb{D}$, and all infinite traces $w \in \Sigma^\omega$, let $\max(\Phi, \Psi)(w) = \max(\Phi(w), \Psi(w))$ and $\min(\Phi, \Psi)(w) = \min(\Phi(w), \Psi(w))$.

Proposition 3.3.7. *For all quantitative properties Φ and Ψ on a lattice, if Φ and Ψ are universally (resp. existentially; approximately) monitorable from below (resp. above), then the property $\max(\Phi, \Psi)$ (resp. $\min(\Phi, \Psi)$) is also universally (resp. existentially; approximately) monitorable from below (resp. above).*

Proof. Let ν_Φ and ν_Ψ be two verdict functions that universally monitor Φ and Ψ from below. Then, we have $\max(\Phi(w), \Psi(w)) = \max(\limsup \nu_\Phi(w), \limsup \nu_\Psi(w))$ for every $w \in \Sigma^\omega$. Since we assume that the domain contains a greatest element, for every $w \in \Sigma^\omega$, we also have $\max(\limsup \nu_\Phi(w), \limsup \nu_\Psi(w))$ equals $\limsup(\max(\nu_\Phi(w), \nu_\Psi(w)))$. Therefore, we can use $\max(\nu_\Phi, \nu_\Psi)$ as a verdict function to universally monitor $\max(\Phi, \Psi)$ from below the same way ν_Φ and ν_Ψ monitor Φ and Ψ. The case for $\min$ and monitorability from above is symmetric, and the cases for existential and approximate monitoring can be proved similarly. □

Proposition 3.3.8. *For all quantitative properties Φ and Ψ on a lattice, if Φ and Ψ are (universally, existentially, or approximately) monitorable from below (resp. above), the property $\min(\Phi, \Psi)$ (resp. $\max(\Phi, \Psi)$) is approximately monitorable from below (resp. above).*

Proof. Let ν_Φ and ν_Ψ be verdict functions that monitor Φ and Ψ from below, therefore for every infinite trace $w \in \Sigma^\omega$ we have $\limsup \nu_\Phi(w) \leq \Phi(w)$ and $\limsup \nu_\Psi(w) \leq \Psi(w)$. Because $\limsup(\min(\nu_\Phi(w), \nu_\Psi(w))) \leq \min(\limsup \nu_\Phi(w), \limsup \nu_\Psi(w))$ for every $w \in \Sigma^\omega$, we can use $\min(\nu_\Phi, \nu_\Psi)$ as a verdict function to approximately monitor $\min(\Phi, \Psi)$ from below. The case for $\max$ is dual. □

If $\mathbb{D}$ is a numerical value domain with addition and multiplication, such as the reals or integers, or their nonnegative subsets, then not all modalities of monitorability are preserved under these operations. For all quantitative properties Φ and Ψ on $\mathbb{D}$, and all infinite traces $w \in \Sigma^\omega$, let $(\Phi + \Psi)(w) = \Phi(w) + \Psi(w)$ and $(\Phi \cdot \Psi)(w) = \Phi(w) \cdot \Psi(w)$. Since $\limsup$ is subadditive and

submultiplicative while $\liminf$ superadditive and supermultiplicative, one can easily conclude the following.

Proposition 3.3.9. *For all quantitative properties Φ and Ψ on a numerical value domain, if Φ and Ψ are (universally, existentially, or approximately) monitorable from below (resp. above), then $\Phi + \Psi$ and $\Phi \cdot \Psi$ are approximately monitorable from below (resp. above).*

However, monitorability is preserved under any monotonically increasing continuous function on value domains that are totally ordered.

Proposition 3.3.10. *Let $\mathbb{D}$ be a totally-ordered value domain. Consider a quantitative property $\Phi : \Sigma^\omega \to \mathbb{D}$ and a monotonically increasing continuous function $f : \mathbb{D} \to \mathbb{D}$. If Φ is (universally, existentially, or approximately) monitorable from below (resp. above), then so is $f(\Phi)$.*

Proof. Follows from the fact that monotonically increasing continuous functions commute with the limit operations. □

3.3.3 Continuous Quantitative Properties

For this section, we assume that $\mathbb{D}$ is a complete lattice and define *continuous-from-above* and *continuous-from-below* properties on $\mathbb{D}$. Let Φ be a quantitative property and, for every $u \in \Sigma^*$, let $\Phi^+(u) = \sup\{\Phi(uw) \mid w \in \Sigma^\omega\}$. For $w \in \Sigma^\omega$, the function Φ^+ generates an infinite sequence $\Phi^+(w) = (\Phi^+(u_i))_{i \in \mathbb{N}}$ over the prefixes $u_i \prec w$ of increasing length i. Similarly, let $\Phi^-(u) = \inf\{\Phi(uw) \mid w \in \Sigma^\omega\}$ and extend it to generate infinite sequences on infinite traces.

Definition 3.3.11. *A property Φ is* continuous from above *iff for every infinite trace $w \in \Sigma^\omega$, we have $\Phi(w) = \lim \Phi^+(w)$. Moreover, Φ is* continuous from below *iff $\overline{\Phi}$ is continuous from above, or equivalently, iff $\Phi(w) = \lim \Phi^-(w)$ for every $w \in \Sigma^\omega$.*

Intuitively, the continuous-from-above and continuous-from-below properties constitute well-behaved sets of properties in the sense that, to monitor them, there is no need for speculation. For example, considering a continuous-from-above property, the least upper bound can only decrease after reading longer prefixes; therefore, a verdict function monitoring such a property can simultaneously be conservative and precise. We make this connection more explicit and show that continuous-from-above and continuous-from-below properties satisfy the desirable property of being universally monitorable by monotonic verdict functions.

Theorem 3.3.12. *A quantitative property Φ is continuous from above iff Φ is universally monitorable by a monotonically decreasing verdict function.*

Proof. For the *only if* direction, suppose Φ is continuous from above, i.e., $\lim \Phi^+(w) = \Phi(w)$ for every $w \in \Sigma^\omega$. Since $\Phi^+ : \Sigma^* \to \mathbb{D}$ is monotonically decreasing and it converges to the property value for every infinite trace, we can use it as the verdict function to universally monitor Φ.

Now, let ν be a monotonically decreasing verdict function such that $\lim \nu(w) = \Phi(w)$ for all $w \in \Sigma^\omega$. We claim that $\nu(u) \geq \Phi^+(u)$ for all $u \in \Sigma^*$. Suppose towards contradiction that $\nu(u) < \Phi^+(u)$ for some $u \in \Sigma^*$. Since we have either (i) $\Phi^+(u) = \Phi(uw)$ for some

$w \in \Sigma^\omega$, or (ii) for every $w \in \Sigma^\omega$ there exists $w' \in \Sigma^\omega$ such that $\Phi(uw) < \Phi(uw')$, we obtain $\nu(u) < \Phi(uw'')$ for some $w'' \in \Sigma^\omega$. It contradicts the assumption that ν is a monotonically decreasing verdict that universally monitors Φ from below, therefore our claim is correct. Now, observe that $\nu(u) \geq \Phi^+(u)$ for all $u \in \Sigma^*$ implies $\lim \nu(w) \geq \lim \Phi^+(w)$ for all $w \in \Sigma^\omega$. Since ν universally monitors Φ and is monotonic, we get $\Phi(w) \geq \lim \Phi^+(w)$ for all $w \in \Sigma^\omega$. By the definition of Φ^+, we also know that for every property Φ and every infinite trace $w \in \Sigma^\omega$, we have $\lim \Phi^+(w) \geq \Phi(w)$. Therefore, we conclude that $\lim \Phi^+(w) = \Phi(w)$ for all $w \in \Sigma^\omega$, i.e., Φ is continuous from above. □

Combining Theorem 3.3.12 and Definition 3.3.11, we immediately get the following characterization for the continuous-from-below properties.

Corollary 3.3.13. *A quantitative property Φ is continuous from below iff Φ is universally monitorable by a monotonically increasing verdict function.*

Let $\mathbb{D}$ be a numerical domain and recall the maximal response-time property from Example 3.3.1. As we discussed previously, it is universally monitorable by a monotonically increasing verdict function, and therefore continuous from below. By the same token, one can define the *minimal response-time* property, which is continuous from above. However, average response time, which requires a non-monotonic verdict function although it is universally monitorable from above, is neither continuous from above nor below. We also remark that discounted safety and co-safety properties [dAHM03] are continuous from above and continuous from below, respectively. In Section 3.4, we will discuss how these notions relate to safety and co-safety in the boolean setting.

3.4 Monitoring Boolean Properties

3.4.1 Boolean Monitorability as Quantitative Monitorability

Quantitative properties generalize boolean properties. For every boolean property $P \subseteq \Sigma^\omega$, the characteristic function $\tau_P : \Sigma^\omega \to \{\mathtt{F}, \mathtt{T}\}$ is a quantitative property, where $\tau_P(w) = \mathtt{T}$ if $w \in P$, and $\tau_P(w) = \mathtt{F}$ if $w \notin P$. Using this correspondence, we can embed the main boolean notions of monitorability within our quantitative framework. For this, we consider four different boolean value domains:

- $\mathbb{B} = \{\mathtt{F}, \mathtt{T}\}$ such that $\mathtt{F}$ and $\mathtt{T}$ are incomparable.
- $\mathbb{B}_\bot = \mathbb{B} \cup \{\bot\}$ such that $\bot < \mathtt{F}$ and $\bot < \mathtt{T}$.
- $\mathbb{B}_t = \{\mathtt{F}, \mathtt{T}\}$ such that $\mathtt{F} < \mathtt{T}$.
- $\mathbb{B}_f = \{\mathtt{F}, \mathtt{T}\}$ such that $\mathtt{T} < \mathtt{F}$.

Most work in monitorability assumes irrevocable verdicts. On the domains $\mathbb{B}$ and $\mathbb{B}_\bot$, where $\mathtt{T}$ and $\mathtt{F}$ are incomparable, the irrevocability of verdicts corresponds to monotonically increasing verdict functions. For these, positive verdicts in $\mathbb{B}_t$ and negative verdicts in $\mathbb{B}_f$ are also irrevocable. The following observations about verdict functions on boolean domains are useful as well.

Remark 3.4.1. *Let ν be a verdict function on $\mathbb{B}$ or $\mathbb{B}_\bot$. If ν is monotonic, it cannot switch between* T *and* F*, as these values are incomparable. Therefore, ν can monitor only $\emptyset$ and Σ^ω in $\mathbb{B}$. If ν is unrestricted, it can switch between* T *and* F *only finitely often because the* $\limsup$ *and* $\liminf$ *over every verdict sequence must be defined.*

We begin with the classical definition of monitorability for boolean properties [PZ06, BLS11]. Let $P \subseteq \Sigma^\omega$ be a boolean property. A finite trace $u \in \Sigma^*$ *positively* (resp. *negatively*) *determines* P iff for every $w \in \Sigma^\omega$, we have $uw \in P$ (resp. $uw \notin P$). The boolean property P is *classically monitorable* iff for every $u \in \Sigma^*$, there exists $t \in \Sigma^*$ such that ut positively or negatively determines P. This definition coincides with the *persistently informative monitorability* of [AAF+19b]. It is also captured by our definition of existential monitorability by monotonic verdicts on $\mathbb{B}_\bot$.

Proposition 3.4.2. *A boolean property P is classically monitorable iff τ_P is existentially monitorable from below by a monotonically increasing verdict function on $\mathbb{B}_\bot$.*

According to [AAF+19b], a boolean property P is *satisfaction* (resp. *violation*) *monitorable* iff there exists a monitor that reaches a positive (resp. negative) verdict for every $w \in P$ (resp. $w \notin P$). More generally, if monitorability is parameterized by a truth domain as in [FFM12], then violation and satisfaction monitorability correspond to monitorability over $\{\bot, \mathtt{F}\}$ and $\{\bot, \mathtt{T}\}$, and capture exactly the classes of safety and co-safety properties, respectively. In our framework, violation (resp. satisfaction) monitorability is equivalent to universal monitorability by monotonically increasing verdicts on $\mathbb{B}_f$ (resp. $\mathbb{B}_t$), because they require reaching an irrevocable negative (resp. positive) verdict for traces that violate (reps. satisfy) the property.

Theorem 3.4.3 ([FFM12])**.** *A boolean property P is safe (resp. co-safe) iff τ_P is universally monitorable by a monotonically increasing verdict function on $\mathbb{B}_f$ (resp. $\mathbb{B}_t$).*

A boolean property P is *partially monitorable* according to [AAF+19b] iff it is satisfaction or violation monitorable. This corresponds to parametric monitorability over the 3-valued domain $\{\bot, \mathtt{T}, \mathtt{F}\}$, and is equivalent to the union of safety and co-safety [FFM12]. Due to the duality of $\mathbb{B}_f$ and $\mathbb{B}_t$, in our framework, partial monitorability corresponds to universal monitorability by monotonic verdict functions on either of these domains.

Corollary 3.4.4. *A boolean property P is safe or co-safe iff τ_P is universally monitorable by a monotonic verdict function on $\mathbb{B}_t$ (equivalently, on $\mathbb{B}_f$).*

Also defined in [AAF+19b] is the notion of *complete monitorability*, which requires both satisfaction and violation monitorability. It is equivalent to our universal monitorability by monotonic verdict functions on $\mathbb{B}_\bot$, meaning that for every trace $w \in P$ we reach a positive verdict, and for every $w \notin P$, a negative verdict.

Theorem 3.4.5 ([AAF+19b])**.** *A boolean property P is both safe and co-safe iff τ_P is universally monitorable by a monotonically increasing verdict function on $\mathbb{B}_\bot$.*

Based on the idea of revocable verdicts, a 4-valued domain $\{\mathtt{T}, \mathtt{F}, \mathtt{T}_c, \mathtt{F}_c\}$ is also considered in [FFM12], where T and F are still irrevocable but the inconclusive verdict $\bot$ is split into two verdicts $\mathtt{T}_c$ ("currently true") and $\mathtt{F}_c$ ("currently false") for more nuanced reasoning on finite traces. In the universe of ω-regular properties, their monitorability over this domain

corresponds to the class of reactivity properties of the safety-progress hierarchy [CMP93]. In our framework, unrestricted verdict functions provide a similar effect as revocable verdicts. We will show in Theorem 3.4.11 and Example 3.4.12 that unrestricted verdict functions on $\mathbb{B}_\perp$ can existentially monitor reactivity properties and more.

Finally, two weak forms of boolean monitorability defined in [AAF+19b] are *sound monitorability* and *informative monitorability*. While sound monitorability corresponds to approximate monitorability in $\mathbb{B}_\perp$, informative monitorability corresponds to approximate monitorability in $\mathbb{B}_\perp$ by monotonic verdicts but excluding the constant verdict function $\perp$.

3.4.2 Monitoring the Safety-Progress Hierarchy

We first show that some of the modalities of quantitative monitoring are equivalent over boolean domains. Proposition 3.4.6 also indicates some limitations of flat value domains.

Proposition 3.4.6. *Let P be a boolean property and τ_P be the corresponding quantitative property. The following statements are equivalent.*

1. *τ_P is existentially monitorable from below by an unrestricted verdict function on $\mathbb{B}$.*
2. *τ_P is universally monitorable from below by an unrestricted verdict function on $\mathbb{B}$.*
3. *τ_P is universally monitorable from below by an unrestricted verdict function on $\mathbb{B}_\perp$.*

Proof. The key observation for the proofs is that $\mathtt{T}$ and $\mathtt{F}$ are incomparable in $\mathbb{B}$ and $\mathbb{B}_\perp$, as pointed out in Remark 3.4.1.

$(1) \iff (2)$: Let τ_P be existentially monitorable from below by a verdict function ν on $\mathbb{B}$. Since $\mathtt{T}$ and $\mathtt{F}$ are incomparable, for every $w \in \Sigma^\omega$, if $\limsup \nu(w) \leq \tau_P(w)$ then $\limsup \nu(w) = \tau_P(w)$ in domain $\mathbb{B}$. Therefore, ν also universally monitors Φ from below in $\mathbb{B}$. The other direction follows from Definitions 3.2.2 and 3.2.3.

$(2) \iff (3)$: The *only if* direction follows from the fact that $\mathbb{B}_\perp$ is an extension of $\mathbb{B}$ with a least element. For the *if* direction, suppose ν is a verdict function that universally monitors τ_P from below in $\mathbb{B}_\perp$. We construct a verdict function ν' that imitates ν in $\mathbb{B}$ as follows: let $\nu'(u) = \nu(u)$ if $\nu(u) \neq \perp$, and $\nu'(u) = \nu(t)$ otherwise, where t is the longest prefix of u such that $\nu(t) \neq \perp$ (if there is no such prefix, assume w.l.o.g. that $\nu(u) = \mathtt{T}$). Now, let $w \in \Sigma^\omega$ be an infinite trace, and observe that whenever $\nu(w)$ converges, so does $\nu'(w)$. If $\nu(w)$ does not converge, then the subsequential limits must be either (i) $\perp$ and $\mathtt{T}$, or (ii) $\perp$ and $\mathtt{F}$. Suppose (i) is true. Then, there exists a prefix $\hat{u} \prec w$ such that for all $\hat{t} \in \Sigma^*$ satisfying $\hat{u}\hat{t} \prec w$ we have $\nu(\hat{u}\hat{t}) = \perp$ or $\nu(\hat{u}\hat{t}) = \mathtt{T}$. If $\nu(\hat{u}) = \mathtt{T}$, then, by construction, ν' always outputs $\mathtt{T}$ starting from $\hat{u}$. Otherwise, ν' outputs $\mathtt{F}$ until ν outputs $\mathtt{T}$ (which is bound to happen by supposition), and converges to $\mathtt{T}$ afterwards. The case for (ii) is dual. Therefore, for every $w \in \Sigma^\omega$, we have $\limsup \nu'(w) = \limsup \nu(w)$, which means that ν' universally monitors Φ from below. $\square$

Proposition 3.4.7. *For every boolean property P, we have that τ_P is existentially monitorable from below by a monotonically increasing verdict function on $\mathbb{B}_t$ iff τ_P is existentially monitorable from below by a monotonically increasing verdict function on $\mathbb{B}_f$.*

Proof. Suppose τ_P is existentially monitorable from below by a monotonically increasing verdict function ν on $\mathbb{B}_t$. Consider the following verdict function: $\nu'(u) = \mathtt{F}$ if $\nu(u) = \mathtt{F}$ and

$\tau_P(uw) = \mathtt{F}$ for all $w \in \Sigma^\omega$; and $\nu'(u) = \mathtt{T}$ if $\nu(u) = \mathtt{T}$ or $\tau_P(uw) = \mathtt{T}$ for some $w \in \Sigma^\omega$. Notice that for every $u \in \Sigma^*$, if $\nu(u) = \mathtt{T}$ then $\tau_P(uw) = \mathtt{T}$ for all $w \in \Sigma^\omega$, and if $\nu(u) = \mathtt{F}$ then $\tau_P(uw) = \mathtt{F}$ for some $w \in \Sigma^\omega$, or $\nu(ut) = \mathtt{T}$ for some $t \in \Sigma^*$. Therefore, we can equivalently formulate ν' as follows: $\nu'(u) = \mathtt{F}$ if $\tau_P(uw) = \mathtt{F}$ for all $w \in \Sigma^\omega$; and $\nu'(u) = \mathtt{T}$ if $\tau_P(uw) = \mathtt{T}$ for some $w \in \Sigma^\omega$. The verdict function ν' is indeed monotonically increasing. Further, $\limsup \nu'(w) = \mathtt{F}$ implies that there is $u \prec w$ such that $\tau_P(uw') = \mathtt{F}$ for every $w' \in \Sigma^\omega$, which means that for every $w \in \Sigma^\omega$ we have $\limsup \nu'(w) \leq \tau_P(w)$.

Next, we show that for every $u \in \Sigma^*$ there exists $w \in \Sigma^\omega$ such that $\limsup \nu'(uw) = \tau_P(uw)$. Suppose towards contradiction that for some $u \in \Sigma^*$ every $w \in \Sigma^\omega$ gives us $\limsup \nu'(uw) < \tau_P(uw)$, i.e., $\limsup \nu'(uw) = \mathtt{T}$ and $\tau_P(uw) = \mathtt{F}$. Since $\limsup \nu'(uw) = \mathtt{T}$ and ν' is monotonically increasing, we get $\nu'(t) = \mathtt{T}$ for every $t \prec uw$. It means that, by construction, for every $t \prec uw$ there exists $w' \in \Sigma^\omega$ such that $\tau_P(tw') = \mathtt{T}$. However, we get a contradiction since $u \prec uw$ and $\tau_P(uw') = \mathtt{F}$ for every $w' \in \Sigma^\omega$ by supposition. Therefore, we conclude that ν' existentially monitors τ_P from below in $\mathbb{B}_f$. The *if* direction can be proved symmetrically. □

Before we relate various modalities of monitoring and boolean value domains to the rest of the safety-progress classification of boolean properties [CMP93], we discuss how boolean safety and co-safety are special cases of continuous-from-above and continuous-from-below properties from Section 3.3. Consider the value domain $\mathbb{B}_t$, let P be a safety property and τ_P be the corresponding quantitative property. Observe that for every $u \in \Sigma^*$, we have $\tau_P^+(u) = \mathtt{F}$ if u negatively determines P, and $\tau_P^+(u) = \mathtt{T}$ otherwise. Since P is safe, we also have $\tau_P(w) = \lim \tau_P^+(w)$ for every $w \in \Sigma^\omega$, which means that τ_P is continuous from above. Moreover, the inverse $\overline{\tau_P}$ is a continuous-from-below property on $\mathbb{B}_f$, and it still corresponds to the same boolean safety property. Therefore, by Theorem 3.3.12, we get that property P is safe iff $\overline{\tau_P}$ is universally monitorable by a monotonically increasing verdict function on $\mathbb{B}_f$. Similarly, co-safety properties correspond to the continuous-from-below properties on $\mathbb{B}_t$; and thus, they are exactly the properties that are universally monitorable by monotonically increasing verdict functions on $\mathbb{B}_t$.

Positive, finite boolean combinations of safety and co-safety properties are called *obligation* properties [CMP93]. Every obligation property P can be expressed in a canonical conjunctive normal form $\bigcap_{i=1}^{n}(S_i \cup C_i)$ for some positive integer n, where S_i is safe and C_i is co-safe for all $1 \leq i \leq n$. Moreover, an obligation property in conjunctive normal form with $n = k$ is a *k-obligation property*.

We prove that obligation properties are universally monitorable in $\mathbb{B}$, which naturally requires finitely many switches between verdicts $\mathtt{T}$ and $\mathtt{F}$. Moreover, we establish an equivalence between the infinite hierarchy of obligation properties and a hierarchy of verdict functions on $\mathbb{B}$.

Theorem 3.4.8. *A boolean property P is a k-obligation property iff τ_P is universally monitorable by a verdict function on $\mathbb{B}$ that changes its value at most $2k$ times.*

Proof. Suppose P is a k-obligation property, in other words, $P = \bigcap_{i=1}^{k}(S_i \cup C_i)$ for some integer $k \geq 1$ where S_i is safe and C_i is co-safe for each $1 \leq i \leq k$. Consider the following verdict function: $\nu(u) = \mathtt{T}$ if for every $1 \leq i \leq k$ we have u does not negatively determine S_i or u positively determines C_i; and $\nu(u) = \mathtt{F}$ if there exists $1 \leq i \leq k$ such that u negatively determines S_i and u does not positively determine C_i. Note that if a finite trace u positively or negatively determines a boolean property, then so does ut for every finite continuation

t. If P cannot be expressed as a $(k-1)$-obligation property, then there exists a sequence of finite traces $u_1 \prec t_1 \prec \ldots \prec u_k \prec t_k$, w.l.o.g., such that for every $1 \leq i \leq k$ the trace u_i negatively determines S_i, does not negatively determine any S_j for $j > i$, and does not positively determine any C_j for $j \geq i$; and the trace t_i positively determines C_i, does not positively determine any C_j for $j > i$, and does not negatively determine any S_j for $j > i$. This is because otherwise some safety or co-safety properties either cannot be determined, which contradicts the fact that P is an obligation property, or they are determined by the same finite traces, which contradicts the fact that P is not a $(k-1)$-obligation property. Then, the worst case for ν is when P is not $(k-1)$-obligation and it reads t_k above, which forces $2k$ switches. One can verify that ν always converges to the correct property value, i.e., $\lim \nu(w) = \tau_P(w)$ for all $w \in \Sigma^\omega$. Therefore, verdict ν universally monitors τ_P in $\mathbb{B}$.

For the other direction, suppose τ_P is universally monitorable by a verdict function on $\mathbb{B}$ that changes its value at most $2k$ times, and assume towards contradiction that P is not a k-obligation property. In particular, suppose P is an m-obligation property for some $m > k$, which cannot be expressed as a k-obligation, and let $P = \bigcap_{i=1}^{m}(S_i \cup C_i)$ where S_i is safe and C_i is co-safe for each $1 \leq i \leq m$. By the same argument used above, there exist finite traces $u_1 \prec t_1 \prec \ldots \prec u_m \prec t_m$, w.l.o.g., such that for every $1 \leq i \leq m$ the trace u_i negatively determines S_i, does not negatively determine any S_j for $j > i$, and does not positively determine any C_j for $j \geq i$; and the trace t_i positively determines C_i, does not positively determine any C_j for $j > i$, and does not negatively determine any S_j for $j > i$. Assume w.l.o.g. that $\nu(\varepsilon) = \mathtt{T}$. After reading each finite trace described above, ν has to switch its output because otherwise we can construct a trace w such that $\lim \nu(w) \neq \tau_P(w)$. But since ν can only change its value $2k$ times, it immediately yields that ν cannot universally monitor τ_P where P is an m-obligation property for $m > k$. Therefore, P must be a k-obligation property. □

The countable intersection of co-safety properties and the countable union of safety properties, i.e., so-called *response* and *persistence* properties [CMP93], respectively, are also universally monitorable.

Theorem 3.4.9. *A boolean property P is a response property iff τ_P is universally monitorable from below by an unrestricted verdict function on $\mathbb{B}_t$.*

Proof. Suppose P is a response property, i.e., there exists a set $S \subseteq \Sigma^*$ such that for every $w \in \Sigma^\omega$ we have $w \in P$ iff infinitely many prefixes of w belong to S. Let ν be a verdict function as follows: $\nu(u) = \mathtt{T}$ if $u \in S$, and $\nu(u) = \mathtt{F}$ if $u \notin S$. Now, let $w \in \Sigma^\omega$ be a trace. We have $\tau_P(w) = \mathtt{T}$ iff for every $u \prec w$ there exists $t \in \Sigma^*$ such that $ut \prec w$ and $ut \in S$ iff $\limsup \nu(w) = \mathtt{T}$. Therefore, ν universally monitors τ_P from below in $\mathbb{B}_t$.

Now, suppose there exists a verdict function ν that universally monitors τ_P from below in $\mathbb{B}_t$. Because ν is a function on $\mathbb{B}_t = \{\mathtt{T}, \mathtt{F}\}$, there is a set $S \subseteq \Sigma^*$ such that $\nu(u) = \mathtt{T}$ for all $u \in S$, and $v(u) = \mathtt{F}$ for all $u \notin S$. Then, we get that $\limsup \nu(w) = \mathtt{T}$ iff for every $u \prec w$ there exists $t \in \Sigma^*$ such that $ut \prec w$ and $ut \in S$ iff $w \in P$. Observe that the set S is exactly as in the definition of a response property, therefore P is a response property. □

The proof for persistence properties is symmetric.

Theorem 3.4.10. *A boolean property P is a persistence property iff τ_P is universally monitorable from below by an unrestricted verdict function on $\mathbb{B}_f$.*

Positive, finite boolean combinations of response and persistence properties are called *reactivity* properties [CMP93]. We consider existential monitorability in $\mathbb{B}_\bot$ by unrestricted verdict functions, and provide a lower bound.

Theorem 3.4.11. *For every boolean reactivity property P, we have that τ_P is existentially monitorable from below by an unrestricted verdict function on $\mathbb{B}_\bot$.*

Proof. Suppose P is a boolean reactivity property, i.e., $P = \bigcap_{i=1}^{k}(R_i \cup P_i)$ for some $k \geq 1$ where R_i is a response and P_i is a persistence property for every $1 \leq i \leq k$. By Theorem 3.4.9, each τ_{R_i} is universally monitorable from below by a verdict function μ_i on $\mathbb{B}_t$. For each $1 \leq i \leq k$, consider the verdict function ν_i on $\mathbb{B}_\bot$ defined as follows: let $\nu_i(u) = \mathtt{T}$ if $\mu_i(u) = \mathtt{T}$ or u positively determines R_i, let $\nu_i(u) = \mathtt{F}$ if u negatively determines R_i, and $\nu_i(u) = \bot$ otherwise. Note that each ν_i existentially monitors τ_{R_i} from below, and for every $w \in \Sigma^\omega$, if $w \in R_i$ for every $1 \leq i \leq k$, then $w \in P$. Then, we can construct the verdict ν to monitor τ_P: Let $v(\varepsilon) = \mathtt{T}$ and let x be a memory for ν that initially contains ε. On non-empty traces, ν outputs $\bot$ until it observes a trace u such that for every $1 \leq i \leq k$ there exists t_i such that $x \prec t_i \preceq u$ and $\nu_i(t_i) = \mathtt{T}$. When ν reads such a trace u, it outputs $\mathtt{T}$, updates x to store u, and outputs $\bot$ until the next trace that satisfies the condition above. Observe that, for every w, if $\limsup \nu(w) = \mathtt{T}$ then $\tau_P(w) = \mathtt{T}$; and for every u there exists w such that $\limsup \nu(uw) = \tau_P(uw)$ unless R_i is negatively determined for some $1 \leq i \leq k$.

If for some response component R_i is negatively determined, then we can switch to monitor corresponding persistence component instead. Consider the verdict μ_i' for τ_{P_i} on $\mathbb{B}_f$ and construct ν_i' on $\mathbb{B}_\bot$ as follows: let $\nu_i'(u) = \mathtt{F}$ if $\mu_i'(u) = \mathtt{F}$ or u negatively determines P_i, let $\nu_i'(u) = \mathtt{T}$ if u positively determines P_i, and $\nu_i'(u) = \bot$ otherwise. One can verify that for every w, if $\limsup \nu_i'(w) = \mathtt{F}$ then $\tau_P(w) = \mathtt{F}$; and for every u there exists w such that $\limsup \nu_i'(uw) = \tau_P(uw)$ unless P_i is positively determined. Once P_i is positively determined, we know that all possible future traces satisfy $R_i \cup P_i$. Then, we can switch to the previous procedure of monitoring the response components, excluding R_i, and repeat as many times as necessary. □

We now demonstrate that Theorem 3.4.11 is indeed a lower bound for the capabilities of existential monitors in $\mathbb{B}_\bot$.

Example 3.4.12. *Let $P = \bigcup_{i \in \mathbb{N}} R_i$ such that each R_i is a response property. In particular, the property P belongs to the class of $G_{\delta\sigma}$ sets in the Borel hierarchy, which strictly contains the reactivity properties. Moreover, suppose that for some $j \in \mathbb{N}$ property R_j is live. Let μ be a verdict on $\mathbb{B}_t$ for τ_{R_j}. We construct a verdict ν on $\mathbb{B}_\bot$ for τ_P as follows: $\nu(u) = \mathtt{T}$ if $\mu(u) = \mathtt{T}$, and $\nu(u) = \bot$ otherwise. Clearly, for every $w \in \Sigma^\omega$, if $w \in R_j$ then $w \in P$, and thus $\limsup \nu(w) \leq \tau_P(w)$. Moreover, since R_j is live, so is P, i.e., every finite trace u can be extended with some w such that $uw \in P$, and thus $\limsup v(uw) = \tau_P(uw)$. It follows that P is existentially monitorable from below by ν on $\mathbb{B}_\bot$.*

The remaining combinations of monitoring modality and value domain allow us to monitor every boolean property.

Theorem 3.4.13. *For every boolean property P, we have that τ_P is existentially monitorable from below by a monotonically increasing verdict function on $\mathbb{B}_t$.*

Proof. Let ν be a verdict function such that $\nu(u) = \mathtt{T}$ if u positively determines P, and $\nu(u) = \mathtt{F}$ otherwise. Observe that ν is indeed monotonically increasing in $\mathbb{B}_t$, and $\limsup \nu(w) \leq \tau_P(w)$ for every $w \in \Sigma^\omega$. Let $u \in \Sigma^*$ be an arbitrary trace. If $\nu(u) = \mathtt{T}$, then $\tau_P(uw) = \mathtt{T}$ for every continuation $w \in \Sigma^\omega$; otherwise, there exists some $w \in \Sigma^\omega$ such that $\tau_P(uw) = \mathtt{F}$. It implies that for every $u \in \Sigma^*$ there exists $w \in \Sigma^\omega$ such that $\limsup v(uw) = \tau_P(uw)$. Therefore, τ_P is existentially monitorable from below by ν. □

Combining Proposition 3.4.7 and Theorem 3.4.13, we carry this result over to domain $\mathbb{B}_f$.

Corollary 3.4.14. *For every boolean property P, we have that τ_P is existentially monitorable from below by a monotonically increasing verdict function on $\mathbb{B}_f$.*

Tables 3.1 and 3.2 summarize the results of this section. The classes of safety, co-safety, obligation, response, persistence, reactivity, and classically monitorable boolean properties are denoted by $Safe$, CoSafe, Obl, Resp, Pers, React, and Mon, respectively. We note that the upper bound for unrestricted existential monitors on $\mathbb{B}_\bot$ is an open problem.

Table 3.1: Correspondence between classes of boolean properties and universal monitorability.

	Universally monitorable from below	
$\mathbb{D}$	Monotonically increasing	Unrestricted verdict
$\mathbb{B}$	$\emptyset$ or Σ^ω (Rem. 3.4.1)	Obl (Thm. 3.4.8)
$\mathbb{B}_\bot$	Safe ∩ CoSafe (Thm. 3.4.5)	Obl (Prop. 3.4.6 + Thm. 3.4.8)
$\mathbb{B}_t$	CoSafe (Thm. 3.4.3)	Resp (Thm. 3.4.9)
$\mathbb{B}_f$	Safe (Thm. 3.4.3)	Pers (Thm. 3.4.10)

Table 3.2: Correspondence between classes of boolean properties and existential monitorability.

	Existentially monitorable from below	
$\mathbb{D}$	Monotonically increasing	Unrestricted verdict
$\mathbb{B}$	$\emptyset$ or Σ^ω (Rem. 3.4.1)	Obl (Prop. 3.4.6 + Thm. 3.4.8)
$\mathbb{B}_\bot$	Mon (Prop. 3.4.2)	at least Mon ∪ React (Thm. 3.4.11)
$\mathbb{B}_t$	any $P \subseteq \Sigma^\omega$ (Thm. 3.4.13)	any $P \subseteq \Sigma^\omega$ (Thm. 3.4.13)
$\mathbb{B}_f$	any $P \subseteq \Sigma^\omega$ (Cor. 3.4.14)	any $P \subseteq \Sigma^\omega$ (Cor. 3.4.14)

We conclude the section with a simple example that demonstrates the concept of precision in the context of boolean properties.

Example 3.4.15. *Let $P = \Diamond(a \vee b \vee c)$, where $\Diamond$ is the* eventually *operator [PP18], and τ_P be the corresponding quantitative property. Consider the following verdict functions on $\mathbb{B}_t$:*

- $\nu_a(u) = \mathtt{T}$ *iff u contains a,*
- $\nu_{ab}(u) = \mathtt{T}$ *iff u contains a or b,*
- $\nu_{bc}(u) = \mathtt{T}$ *iff u contains b or c,*
- $\nu_{abc}(u) = \mathtt{T}$ *iff u contains a or b or c.*

All these verdict functions are monotonic. Moreover, functions ν_a, ν_{ab}, and ν_{bc} existentially monitor τ_P from below while ν_{abc} monitors universally.

Observe that ν_{ab} is more precise than ν_a, because for every finite prefix u that yields $\nu_a(u) = \mathtt{T}$, we also get $\nu_{ab}(u) = \mathtt{T}$, but not vice versa, considering the traces that contain b but not a. However, we cannot compare ν_{ab} and ν_{bc}, as for every $u \prec a^\omega$, we have $\nu_{bc}(u) \leq \nu_{ab}(u)$ and $\limsup \nu_{bc}(a^\omega) \leq \limsup \nu_{ab}(a^\omega)$, and vice versa for c^ω. Finally, ν_{abc} is the most precise among these verdicts as it universally monitors τ_P.

3.5 Approximate Register Monitors

3.5.1 Verdicts Generated by Register Machines

In this section, we follow [FHS18, FHK20] to define *register machines* as a model for generating an output stream that represents a verdict sequence for monitoring quantitative properties. We consider a set of integer-valued *registers* denoted X. A *valuation* $\mathsf{v} : X \to \mathbb{Z}$ is a mapping from the set of registers to integers. An *update* is a function from valuations to valuations, and a *test* is a function from valuations to $\mathbb{B}$. The set of updates over X is denoted by $\Gamma(X)$, and the set of tests by $\Pi(X)$. We describe updates and tests over X using integer- and boolean-valued expressions, called *instructions*.

Definition 3.5.1. *A (deterministic)* register machine *is a tuple $M = (X, Q, \Sigma, \Delta, q_0, \mathbb{D}, \lambda)$, where X is a finite set of registers, Q is a finite set of states, Σ is a finite alphabet, $\Delta \subseteq Q \times \Sigma \times \Pi(X) \times \Gamma(X) \times Q$ is a set of edges, $q_0 \in Q$ is the initial state, $\mathbb{D}$ is an output value domain, and $\lambda : Q \times \mathbb{Z}^X \to \mathbb{D}$ is an output function such that for every state $q \in Q$, letter $\sigma \in \Sigma$, and valuation v, there is exactly one outgoing edge $(q, \sigma, \phi, \gamma, q') \in \Delta$ with $\mathsf{v} \models \phi$.*

Let $M = (X, Q, \Sigma, \Delta, q_0, \mathbb{D}, \lambda)$ be a register machine. A pair consisting of a state $q \in Q$ and a valuation $\mathsf{v} : X \to \mathbb{Z}$ constitute a *configuration* of M. The initial configuration (q_0, v_0) of M is such that $\mathsf{v}_0(x) = 0$ for every $x \in X$. Between two configurations of M, the *transition* relation is defined by $(q, \mathsf{v}) \xrightarrow{\sigma} (q', \mathsf{v}')$ iff there exists an edge $(q, \sigma, \phi, \gamma, q') \in \Delta$ such that $\mathsf{v} \models \phi$ and $\mathsf{v}' = \gamma(\mathsf{v})$. On an infinite word $w = \sigma_1\sigma_2\ldots$, the machine M produces an infinite sequence of transitions $(q_0, \mathsf{v}_0) \xrightarrow{\sigma_1} (q_1, \mathsf{v}_1) \xrightarrow{\sigma_2} \cdots$, and an infinite *output sequence* $(\lambda(q_i, \mathsf{v}_i))_{i\in\mathbb{N}}$.

Definition 3.5.2. *A register machine $M = (X, Q, \Sigma, \Delta, q_0, \mathbb{D}, \lambda)$* generates *the verdict function $\nu : \Sigma^* \to \mathbb{D}$ iff for every finite trace $u \in \Sigma^*$ the machine M after reading u reaches a configuration (q, v) such that $\lambda(q, \mathsf{v}) = \nu(u)$.*

We mainly focus on a simple form of register machines which can only increment, reset, and compare registers. The according instruction set is denoted by $\langle 0, +1, \geq\rangle$, and expressively equivalent to the instruction set $\langle 0, +1, -1, \geq 0\rangle$, as was shown in [FHS18].

Definition 3.5.3. *A* counter machine *is a register machine with the instructions $x \leftarrow 0$, $x \leftarrow x+1$, and $x \geq y$ for registers $x, y \in X$ and an output function that in every state outputs 0, ∞, or one of the register values. A verdict function ν is a* k-counter verdict function *iff ν is generated by a counter machine M with k registers. A quantitative property Φ is* k-counter monitorable *iff there is a k-counter verdict function that monitors Φ.*

Note that we can use the various modalities of monitoring defined in Section 3.2. For instance, a property Φ is existentially k-counter monitorable from below iff Φ is k-counter monitorable and the witnessing verdict function existentially monitors Φ from below.

One can also define *extended counter machines* with generic output functions. For example, a verdict function generated by an extended 3-counter machine (with an output function that can perform division) can universally monitor the average response-time property from above, as demonstrated in Example 3.3.3.

We remark that our model of register machines is more general than register transducer models operating over uninterpreted infinite alphabets, which typically cannot count (see, e.g., [KMB18]).

3.5.2 Precision-Resource Tradeoffs for Register Machines

In the following example, we illustrate how the arithmetic operations of register machines can play a role in precision-resource tradeoffs for monitoring.

Example 3.5.4 (Adders versus counters)**.** *Let $\Sigma = \{a, b\}$ and $\mathbb{D} = \overline{\mathbb{N}}$. Let Φ be a property such that $\Phi(w) = 2^n$, where n is the length of the longest uninterrupted sequence of a's in $w \in \Sigma^\omega$ if it is bounded, and $\Phi(w) = \infty$ otherwise. Consider a 2-register machine M with the following instructions: $x \leftarrow 1$, $x \leftarrow x + y$, and $x \geq y$ for $x, y \in X$. When M starts reading a segment of a's, it resets one of its registers, say x, to 1 and doubles its value after each a. After the segment ends, it compares the value of x with the other register, say y, and stores the maximum in y, which determines the output value. This way M can generate ν_{add} such that $\nu_{add}(u) = 2^n$, where n is the length of the longest uninterrupted sequence of a's in $u \in \Sigma^*$. Verdict ν_{add} is monotonically increasing and it universally monitors Φ. Now, suppose that we have, instead, a verdict ν_{count} that is generated by a 2-counter machine. Since the counter values can only grow linearly, we can have $\nu_{count}(u) = 2n$ for n as above. Although it grows much slower, ν_{count} existentially monitors Φ from below, because the extension a^ω yields $\limsup \nu_{count}(sa^\omega) = p(sa^\omega) = \infty$ for every $u \in \Sigma^*$. Since ν_{add} universally monitors Φ, it is clearly more precise than ν_{count}.*

Recall the two-pair maximal response-time property from Section 3.1. We can generalize this property to give an example for a precision-resource tradeoff on the number of counter registers that are available for monitoring.

Example 3.5.5 (Counter machine)**.** *Let $k \in \mathbb{N}$ and let $\Sigma_k = \{\mathsf{rq}_1, \mathsf{ack}_1, \ldots, \mathsf{rq}_k, \mathsf{ack}_k, \mathsf{oo}\}$. The k-pair maximal response-time property $\Phi : \Sigma^\omega \to \overline{\mathbb{N}}^k$ specifies the maximal response times for all $(\mathsf{rq}, \mathsf{ack})$ pairs in Σ_k. More explicitly, for every $1 \leq i \leq k$, let Φ_i specify the maximal response-time for pair $(\mathsf{rq}_i, \mathsf{ack}_i)$ as in Example 3.3.1, and let $\Phi(w) = (\Phi_1(w), \ldots, \Phi_k(w))$ for every $w \in \Sigma^\omega$. As hinted in Section 3.1, there is a $2k$-counter verdict function ν_{2k} which simply combines the 2-counter verdict functions μ_i that universally monitor Φ_i for all $1 \leq i \leq k$. Specifically, $\nu_{2k}(u) = (\mu_1(u), \ldots, \mu_k(u))$ for every $u \in \Sigma^*$.*

Observe that a $(k+1)$-counter verdict function ν_{k+1} cannot universally monitor Φ, because whenever it reads a trace that contains more than one active request, it needs to either ignore some active requests and process only one of them, or forget the maximal response time for some pairs and use those counters to process the active requests. However, it can existentially monitor Φ from below, because every finite trace can be extended with a continuation w in which all previously active requests are acknowledged and the true maxima occur in w one

by one. If the server has at most one active request at any given time, then $k+1$ counters suffice for universal monitoring. This is because it only needs to use one register to process the current response time while storing the maxima in the remaining k counters. Let Φ' be the variant of Φ under the assumption of no simultaneous requests, and suppose we have a $(\frac{k}{2}+1)$-counter verdict function $\nu_{\frac{k}{2}+1}$. By the same reason that ν_{k+1} cannot universally monitor Φ, the function $\nu_{\frac{k}{2}+1}$ cannot universally monitor Φ'. However, for every odd number $1 \leq i < k$, we can assign one counter to store $\max(\mu_i(u), \mu_{i+1}(u))$, which provides an over-approximation for either Φ_i or Φ_{i+1} while being precise for the other. Overall, although it can provide a meaningful approximation, the function $\nu_{\frac{k}{2}+1}$ is less precise than ν_{k+1}.

The following theorems generalize this example.

Theorem 3.5.6. *For every $k > 1$, there exists a quantitative property Φ_k such that Φ_k is universally monitorable by a monotonically decreasing k-counter verdict function ν_k. Moreover, for every $\ell < k$ and every monotonically decreasing ℓ-counter verdict function ν_ℓ that approximately monitors Φ_k from below (resp. above), there exists a monotonically decreasing $(\ell+1)$-counter verdict function $\nu_{\ell+1}$ that approximately monitors p_k from below (resp. above) such that $\nu_{\ell+1}$ is more precise than ν_ℓ.*

Proof. For convenience, we consider the $\langle 0, +1, -1, \geq 0\rangle$ variant of counter machines. Let $\Sigma_k = \{1, \ldots, k\}$. For every $u \in \Sigma_k^*$ and $i \in \Sigma_k$ denote by $|u|_i$ the number of occurrences of the letter i in u. Consider the boolean safety property $P_k = \{w \in \Sigma_k^\omega \mid \forall\, 1 \leq i < k : \forall u \prec w : |u|_i \geq |u|_{i+1}\}$, and let Φ_k be as follows: $\Phi_k(w) = \infty$ if $w \in P_k$, and $\Phi_k(w) = |t|$ otherwise, where t is the shortest prefix of w that negatively determines P_k. We construct a verdict function ν_k as follows: $\nu_k(u) = \infty$ if u does not negatively determine P_k, and $\nu_k(u) = |t|$ otherwise, where t is the shortest prefix of u that negatively determines P_k. The verdict ν_k is monotonically decreasing and it can be generated by a k-counter machine where, for every $1 \leq i < k$ and $u \in \Sigma_k^*$, the counter x_i stores $|u|_i - |u|_{i+1}$, and x_k stores $|u|$. Moreover, because we need $k-1$ counters to recognize P_k (see Thm. 4.3 in [FHS18]) and one more to store the output, Φ_k is not universally ℓ-counter monitorable for $\ell < k$.

Let $\ell < k$, and take a monotonically decreasing ℓ-counter verdict function ν_ℓ that approximately monitors Φ_k from below. Note that, for every $u \in \Sigma_k^*$, if the generating counter machine does not store a linear function $\alpha(u) \leq |u|$, then ν_ℓ can be either the constant 0 function or a function that switches from ∞ to 0 and never misses a violation. Then, we construct an $(\ell+1)$-counter machine that stores $|u|$ in the new counter. If ν_ℓ is constant, it uses the rest to count $|u|_i - |u|_{i+1}$ for $1 \leq i < \ell$ and catch violations, similarly as ν_k above; otherwise, it outputs $|u|$ instead of 0. The resulting verdict $\nu_{\ell+1}$ is monotonically decreasing and approximately monitors p_k from below. It is also more precise than ν_ℓ. Now, suppose the generating ℓ-counter machine counts a linear function $\alpha(u) \leq |u|$. Since this machine cannot recognize P_k, there exists $w \in P_k$ such that $\lim \nu_\ell(w) < \infty$, i.e., ν_ℓ incorrectly concludes that $|u|_i < |u|_{i+1}$ for some $u \prec w$ and $1 \leq i < k$. We construct an $(\ell+1)$-counter machine M where the additional counter keeps track of $|u|_i - |u|_{i+1}$ for every $u \in \Sigma_k^*$, the output register stores $|u|$, and the rest behave the same as in ν_ℓ. Moreover, whenever ν_ℓ concludes that $|u|_i < |u|_{i+1}$, the behavior of M is determined by the correct value of $|u|_i - |u|_{i+1}$ stored in the new counter. It yields that the verdict $\nu_{\ell+1}$ generated by M is more precise than ν_ℓ because $\lim \nu_\ell(w) < \lim \nu_{\ell+1}(w)$ for some trace $w \in \Sigma^\omega$. The case for monitoring from above is similar. □

Since monitoring Φ_k in the proof above involves recognizing a boolean property P_k, the counter machine for Φ_k must be able to distinguish traces with respect to P_k. To achieve this, intuitively, the machine needs a counter for each "independent" quantity. Also, the use of a variable-size alphabet Σ_k is merely a convenience. We can encode every word over Σ_k in binary with the help of an additional separator symbol. More explicitly, we can take a ternary alphabet $\Sigma = \{0, 1, \#\}$ to represent every letter in Σ_k as a binary sequence and separate the sequences by $\#$. We combine these observations to construct a quantitative property for which counting does not suffice for universal monitoring no matter the number of registers, but each additional register gives a better approximation.

Theorem 3.5.7. *There exists a quantitative property Φ such that for every $k > 1$ and every k-counter verdict function ν_k that approximately monitors Φ from below (resp. above), there exists a $(k+1)$-counter verdict function ν_{k+1} that approximately monitors Φ from below (resp. above) and is more precise than ν_k.*

Proof. Let $\Sigma = \{0, 1, \#\}$. For every $u \in \Sigma^*$ and $i \in \mathbb{N}$, let $n_i(u)$ denote the number of occurrences of the binary sequence that corresponds to i in the longest prefix of u that ends with $\#$. For example, if we have $u = 001\#10$, then $n_1(u) = 1$ and $n_2(u) = 0$. Similarly as in the proof of Theorem 3.5.6, consider counter machines with instructions $\langle 0, +1, -1, \geq 0\rangle$ and the following boolean safety property: $P = \{w \in \Sigma^\omega \mid \forall i \in \mathbb{N} : \forall u \prec w : n_i(u) \geq n_{i+1}(u)\}$. We define the quantitative property Φ as $\Phi(w) = \infty$ if $w \in P$, and $\Phi(w) = |t|_\#$ where t is the shortest prefix of w that negatively determines P. Observe that P is a generalization of P_k in the proof of Theorem 3.5.6, and it requires counting infinitely many distinct quantities. Therefore, one can show that, to universally monitor Φ, one needs infinitely many counter registers. However, for every $k > 1$, there exists a k-counter verdict function ν_k that approximately monitors Φ from below or above, for instance, by keeping track of $n_i(u) - n_{i+1}(u)$ for every $1 \leq i < k$ and counting $\#$'s in the remaining register.

We now construct a $(k+1)$-counter verdict function ν_{k+1} from ν_k. Suppose ν_k approximately monitors Φ from below. Note that the generating machine of ν_k lacks the resources to distinguish traces with respect to P correctly. Therefore, regardless of the monotonicity of ν_k, a similar reasoning as in the proof of Theorem 3.5.6 applies. We can use the additional counter of ν_{k+1} to keep track of $n_i(u) - n_{i+1}(u)$ or $|u|_\#$ for every $u \in \Sigma^*$ while the rest operate the same as in ν_k. It yields that $\limsup \nu_k(w) < \limsup \nu_{k+1}(w)$; thus ν_{k+1} is more precise than ν_k. One can similarly show the case for monitoring from above. □

3.6 Conclusion

We argued for the need of a quantitative semantic framework for runtime verification which supports monitors that over- or under-approximate quantitative properties, and we provided such a framework.

An obvious direction for future work is to systematically explore precision-resource tradeoffs for different monitor models and property classes. For example, a quantitative property class that we have not considered in this work is the limit monitoring of statistical indicators [FHK20]. Other interesting resources that play a role in precision-resource tradeoffs are the "speed" or rate of convergence of monitors, that is, how quickly a monitor reaches the desired property value, and "assumptions", that is, prior knowledge about the system or the environment that can be used by the monitor [HS20, AAF+21a]. We also plan to consider the reliability of communication channels [KHF20] and how it relates to monitoring precision.

Another question is the synthesis problem: given a quantitative property Φ and a register machine template (instruction set and number of registers), does there exist a register machine M generating a verdict function ν that universally or existentially monitors Φ from below or above?

Building on our definitions of continuous-from-above and continuous-from-below quantitative properties, one can define a generalization of the safety-progress hierarchy [CMP93] to obtain a Borel classification of quantitative properties.

Lastly, a logical extension of monitoring is *enforcement* [LR10, FMFR11, FFM12], that is, manipulating the observed system's behavior to prevent undesired outcomes. We aim to extend the notion of enforceability from the boolean to the quantitative setting and explore precision-resource tradeoffs for enforcement monitors (a.k.a. *shields* [KAB+17]).

CHAPTER 4

Abstract Monitors for Quantitative Properties

In this chapter, the following publications were re-used in full:

- Thomas A. Henzinger, Nicolas Mazzocchi, N. Ege Saraç. *Abstract Monitors for Quantitative Specifications*. In Runtime Verification - 22nd International Conference, **RV 2022**.

4.1 Introduction

Online monitoring is a runtime verification (RV) technique [BFFR18] that, by sacrificing completeness, aims to lighten the burden caused by exhaustive formal methods. A monitor watches an unbounded sequence w of observations, called trace, one observation at a time. At each time $n \geq 0$, it tries to provide information about the value assigned to w by the property. For a boolean property P, after each trace prefix u, the monitor may output one of three values: all infinite extensions of u satisfy P, violate P, or neither [BLS11].

Quantitative properties [CDH10b] generalize their boolean analogs by assigning each trace w a value from some richer domain. For example, the boolean property Resp assigns *true* to w iff every observation `rq` in w is eventually followed by an observation `ack` in w, while the quantitative property MaxRespTime assigns the least upper bound on the number of observations between each `rq` and the corresponding `ack`, or ∞ if there is no such upper bound.

In the limit monitoring of a quantitative property Φ over a trace w, a limit (e.g., $\limsup$, $\liminf$) of the infinite sequence of monitor outputs should provide information about the value $\Phi(w)$ assigned to the trace. For example, a "natural way to monitor" MaxRespTime is to have the monitor output, at each time, the maximum of (i) the maximal response time so far and (ii) the time since the least recent pending `rq`, if there is a pending `rq`. The $\limsup$ (and $\liminf$) of this infinite output sequence converges towards MaxRespTime.

In contrast to its boolean analog, the quantitative setting naturally supports approximation. A monitor has error $\delta \geq 0$ if, for all infinite traces, the limit of the output sequence is within δ of the property value. In particular, this leads to precision-resource tradeoffs for quantitative

monitors: The provisioning of additional states, registers, or operations may reduce the error, and a larger error tolerance may enable monitors that use fewer resources.

In this chapter, we provide a formal framework for studying such precision-resource tradeoffs for an abstract definition of quantitative monitors. This abstract framework can be instantiated, for example, by finite-state monitors or register monitors, where a finite-state monitor remembers a bounded amount of information about each trace prefix, and a register monitor remembers a bounded number of integer values [FHS18]. For us, an *abstract monitor* partitions, at each time n, all prefixes of length up to n into a finite number of equivalence classes such that if two prefixes u_1 and u_2 are equivalent, then the monitor outputs the same value after observing u_1 and u_2. The number of equivalence classes introduced at time n provides a natural measure for the resource use of the abstract monitor after n observations.

In this setting, where the *resource use* of a monitor is measured, we also want to measure the *precision* of a monitor. To define the precision of a monitor after a finite trace prefix, we need to enrich our definition of quantitative properties: We let a quantitative property assign values not only to infinite traces but also to finite traces. Indeed, many property values for infinite traces are usually defined as limits [HS21]. For example, what we called above the "natural way to monitor" MaxRespTime using two counters is, in fact, the usual formal definition of the quantitative property MaxRespTime.

Once both properties and monitors assign values to all finite traces, there is a natural definition for the precision of a monitor: At each time n, the *prompt error* is the maximal difference between the monitor output and the property value over all finite traces of length up to n. Furthermore, the *limit error* is the least upper bound on the difference between the limit of monitor outputs and the limit of property values over all infinite traces. Note that if the prompt error of a monitor is 0, then so is the limit error, but not necessarily vice versa. An exact-value monitor (i.e., a monitor with prompt error $\delta = 0$) implements the property as it is defined. In contrast, an approximate monitor (i.e., a monitor with prompt error $\delta > 0$) of the same property may use fewer resources. An approximate monitor may still achieve limit error 0, which is a situation of particular interest that we study.

Given an abstract monitor, one way to obtain a new monitor that uses fewer resources use is to merge some equivalence classes, and one way to increase the precision is to split some equivalence classes. However, this naive approach toward reaching a desired precision or resource use is not always the best. For an approximate monitor with a given prompt error and limit error, the goal is *resource optimality*, i.e., minimizing the resource use as much as the error threshold allows. We will see that merging the equivalence classes of a given monitor may not yield a resource-optimal one.

The limit error of a monitor is bounded by its prompt error. We also investigate the case where we require a certain limit error while leaving the prompt error potentially unbounded. We will see that allowing arbitrary prompt error may not permit the monitor to save resources if the desired limit error is fixed. We say that such properties have *resource-intensive limit behavior*. In fact, MaxRespTime displays resource-intensive limit behavior. Other examples include a subclass of *reversible properties*. Reversibility is a notion from automata theory characterized by the property being realizable with a finite-state automaton that is both forward and backward deterministic. A similar notion, generalized to the quantitative setting, can be introduced in our framework, allowing an abstract monitor to process an infinite trace in a two-way fashion.

Overview. Section 4.2 formalizes the framework of abstract monitors and provides insights on relations between basic notions such as resource use and precision.

Section 4.3 focuses on monitoring with bounded error over finite traces. First, in Subsection 4.3.1, we show that the exact-value monitor over finite traces is unique and resource optimal for every property. Additionally, for resource-optimal approximate monitors, we prove: (i) they are not unique in Subsection 4.3.1, (ii) they do not necessarily follow the structure of the exact-value monitor in Subsection 4.3.2, and (iii) they do not necessarily minimize their resource use at each time in Subsection 4.3.2. Then, in Subsection 4.3.3, we study the precision-resource tradeoff suitability: We exhibit (i) a property for which we can arbitrarily improve the resource use by damaging precision, and (ii) another for which we arbitrarily improve the precision by damaging the resource use.

Section 4.4 focuses on monitoring without error on infinite traces. In particular, in Subsection 4.4.1 we provide a condition for identifying properties with resource-intensive limit behavior, for which having zero limit error prevents the tradeoff between resource use and error on finite traces. This condition captures two paradigmatic properties: (i) maximal response time and (ii) average response time. Finally, in Subsection 4.4.2 we investigate reversible properties, which can be implemented in a manner both forward and backward deterministic. A subclass of reversible properties have resource-intensive limit behavior, which we demonstrate through the average ping property.

Section 4.5 concludes the chapter and addresses future research directions our framework offers.

Related work. In the boolean setting, several notions of monitorability have been proposed over the years [BLS11, FFM12, FAA$^+$17]. Much of the theoretical efforts have focused on regular properties [AAF$^+$21b, BLS10, MB15], although some proposed more expressive models [BFH$^+$12, BKMZ15, dSS$^+$05]. We refer the reader to [BFFR18] for coverage of these and more.

Verification of quantitative system properties have received significant attention, both in the deterministic [CDH10b, DDG$^+$10, CDE$^+$10, HPPR18] and the probabilistic setting [Kwi07, BCFK15, CD11, FKN$^+$11] In the context of RV, the literature on properties with quantitative aspects features primarily metric temporal logic and signal temporal logic [HOW14, JBG$^+$18, MN04, MCW21a, MCW21b]. Other efforts include processing data streams with a focus on deciding their properties at runtime [AMS17, AMS19] and an extension of weighted automata with monitor counters [CHO16]. None of these works focus on monitoring quantitative properties with approximate verdicts or the relation between monitorability and monitor resources.

Approximate methods have long been used in verification and related areas [Cou96, HS00, Alb03], including distributed computation of aggregate functions [CLKB04, SBAS04, SBY06] and approximate determinization or minimization of quantitative models [AKL13, BH12, HK12]. To the best of our knowledge, the use of approximate methods in monitoring mainly concentrates on the property rather than taking approximateness as a monitor feature and studying the quality of monitor verdicts. In predictive or assumption-based monitoring [CTT21, ZLD12] and for monitoring hyperproperties [SSSB21], an over-approximation of the system under observation is used as an assumption to limit the set of possible traces [HS20]. Similarly, in runtime quantitative verification [CGJP13, NKF20], the underlying probabilistic model of the system is approximated and continually updated. For monitoring under partial observability, [ADL14]

describes an approach to approximate the given property for minimizing the number of undetected violations. In the branching-time setting, [AAF+21a] uses a monitorable under- or over-approximation of the given property to construct an "optimal" monitor. Nonetheless, a form of distributed and approximate limit monitoring for spatial properties was studied in [ACD+21]. None of these works consider approximateness as a monitor property to study the relation between monitor resources and the quality of its verdicts.

Recently, [FHS18] introduced a concrete monitor model with integer-valued registers and studied their resource needs. This model was later used for limit monitoring of statistical indicators of traces under probabilistic assumptions [FHK20]. A general framework for approximate limit monitoring of quantitative properties was proposed in [HS21]. However, that framework focuses exclusively on limit behaviors and on specific monitor models such as finite automata and register machines, thus allowing only limited precision-cost analyses. The main innovations of the present work over previous work are twofold. First, we abstract the monitor model and its resource use away from specific machine models. Second, by introducing prompt errors, we study the resource use of monitors over time and relate this to the monitoring precision over time. This more nuanced framework enables a more fine-grained analysis and comparison of different monitors for the same property concerning their precision and resource use.

4.2 Definitional Framework

Recall that a *value domain* $\mathbb{D}$ is a complete lattice, unless stated otherwise, and a *quantitative property* $\Phi : \Sigma^\omega \to \mathbb{D}$ is a total function from infinite traces to values. In this chapter, we focus on quantitative properties whose value domain is $\overline{\mathbb{R}}$ or a bounded subset thereof.

A binary relation $\sim$ over Σ^* is an *equivalence relation* when it is reflexive, symmetric, and transitive. For a given equivalence relation $\sim$ over Σ^* and a finite trace $u \in \Sigma^*$, we denote by $[u]_\sim$ the equivalence class of $\sim$ in which u belongs. When $\sim$ is clear from the context, we write $[u]$ instead. A *right-monotonic* relation $\sim$ over Σ^* fulfills $u_1 \sim u_2 \Rightarrow u_1 t \sim u_2 t$ for all $u_1, u_2, t \in \Sigma^*$.

Throughout the chapter, we use $\Box$ and $\Diamond$ to denote the linear temporal logic (LTL) operators *always* and *eventually*, respectively. See [PP18] for interpretation of LTL operators on infinite traces, and [GV13, CMP93, EFH+03] on finite traces.

4.2.1 Limit Properties

In this chapter, we focus on properties that can be defined as limits of value sequences. Let us recall below the definition of limit properties capturing this idea.

A *finitary property* $\pi \colon \Sigma^* \to \mathbb{D}$ associates a value with each finite word. A *value function* $\mathsf{Val} \colon \mathbb{D}^\omega \to \mathbb{D}$ accumulates an infinite sequence of values to a single value. Given an infinite sequence of values $x = x_1 x_2 \ldots$, we write $\mathsf{LimInf}(x) = \lim_{n \mapsto +\infty} \inf\{x_i \mid i \geq n\}$ and $\mathsf{LimSup}(x) = \lim_{n \mapsto +\infty} \sup\{x_i \mid i \geq n\}$. Whenever $\mathsf{LimInf}(x) = \mathsf{LimSup}(x)$ for a given sequence x, we simply write $\mathsf{Lim}(x)$ for its value.

A property $\Phi : \Sigma^\omega \to \mathbb{D}$ is a *limit property* when there exists a finitary property $\pi : \Sigma^* \to \mathbb{D}$ and a value function $\mathsf{Val} : \mathbb{D}^\omega \to \mathbb{D}$ such that $\Phi(w) = \mathsf{Val}_{u \prec w} \pi(u)$ for all $w \in \Sigma^\omega$. We denote this by $\Phi = (\pi, \mathsf{Val})$, write $\Phi(u)$ instead of $\pi(u)$ when $u \in \Sigma^*$, and call Φ a Val-*property*.

Together with a given property Φ, we define the right-monotonic equivalence relation $\sim^*_\Phi$ as follows. For all $u_1, u_2 \in \Sigma^*$ we have $u_1 \sim^*_\Phi u_2$ iff $\pi(u_1 t) = \pi(u_2 t)$ holds for all $t \in \Sigma^*$.

We define below the *discounted response* property. Throughout the section, we will use this property as a running example.

Example 4.2.1. *Let $\Sigma = \{rq, ack, oo\}$ and consider the LTL response property $P = \square(rq \to \Diamond ack)$. Let $0 < \lambda < 1$ be a discount factor. We define* $\mathsf{DiscResp}(u) = 1$ *if $u \in P$, and* $\mathsf{DiscResp}(u) = \lambda^n$ *otherwise, where $n = |u| - |t|$ and $t \prec u$ is the longest prefix of t with $t \in P$. We define $\Phi_{\mathsf{DR}} = (\mathsf{DiscResp}, \mathsf{LimSup})$, the* discounted response *property. Intuitively, Φ_{DR} assigns each finite trace a value that shows how close the system behaves to P such that, at the limit, it denotes whether the infinite behavior satisfies P or not.*

Now, take two traces $u_1, u_2 \in \Sigma^*$. We claim that $u_1 \sim^*_{\Phi_{\mathsf{DR}}} u_2$ iff either (i) both traces have no pending request or (ii) both have a request pending for the same number of steps. First, we assume $u_1 \sim^*_{\Phi_{\mathsf{DR}}} u_2$ holds and note that we must have $\Phi_{\mathsf{DR}}(u_1 t) = \Phi_{\mathsf{DR}}(u_2 t)$ for every $t \in \Sigma^*$. Then, we eliminate the cases other than (i) and (ii) as follows. If, w.l.o.g., $u_1 \in P$ and $u_2 \notin P$, then $\Phi_{\mathsf{DR}}(u_2) < \Phi_{\mathsf{DR}}(u_1) = 1$, thus $u_1 \not\sim^*_{\Phi_{\mathsf{DR}}} u_2$. If, w.l.o.g., u_1 has a request pending for i steps and u_2 for $j > i$ steps, then $\Phi_{\mathsf{DR}}(u_2) = \lambda^j < \lambda^i = \Phi_{\mathsf{DR}}(u_1)$, thus $u_1 \not\sim^*_{\Phi_{\mathsf{DR}}} u_2$. The other direction is similar.

4.2.2 Abstract Monitors

We are now ready to present our abstract definition of quantitative monitors.

Definition 4.2.2 (Monitor)**.** *A* monitor *$\mathcal{M} = (\sim, \gamma)$ is a tuple consisting of a right-monotonic equivalence relation $\sim$ on Σ^* and a function $\gamma\colon (\Sigma^*/\sim) \to \overline{\mathbb{R}}$. Let $\delta_{\mathsf{fin}}, \delta_{\lim} \in \overline{\mathbb{R}}$ be error thresholds. We say that $\mathcal{M}$ is a* $(\delta_{\mathsf{fin}}, \delta_{\lim})$-monitor *for a given property $\Phi = (\pi, \mathsf{Val})$ iff*

- $|\pi(u) - \gamma([u])| \leq \delta_{\mathsf{fin}}$ *for all $u \in \Sigma^*$, and*
- $|\mathsf{Val}_{u \prec w}(\pi(u)) - \mathsf{Val}_{u \prec w}(\gamma([u]))| \leq \delta_{\lim}$ *for all $w \in \Sigma^\omega$.*

We say that $\mathcal{M}$ has a prompt error *of δ_{fin} and a* limit error *of $\delta_{\lim}$.*

In the sequel, we write as shorthand $\pi(w)$ for $(\pi(u_i))_{i \in \mathbb{N}}$ and $\gamma([w])$ for $(\gamma([u_i]))_{i \in \mathbb{N}}$ for every $w \in \Sigma^\omega$ and its prefixes $u_i \prec w$ of increasing length i. We further write $\mathcal{M}(u) = \gamma([u])$ when $u \in \Sigma^*$ and $\mathcal{M}(w) = \mathsf{Val}(\gamma([w]))$ when $w \in \Sigma^\omega$.

Observe that for every property there is an obvious monitor that imitates exactly the property, which we define as follows.

Definition 4.2.3 (Exact-value monitor)**.** *Let $\Phi = (\pi, \mathsf{Val})$ be a property. The* exact-value monitor *of Φ is defined as $\mathcal{M}_\Phi = (\sim^*_\Phi, u \mapsto \pi(u))$.*

A monitor for a given property is *approximate* when it differs from the property's exact-value monitor. Below, we demonstrate the exact-value monitor and an approximate monitor for the discounted response property.

Example 4.2.4. *Recall from Example 4.2.1 the discounted response property Φ_{DR}. Clearly, its exact-value monitor is $\mathcal{M}_{\Phi_{\mathsf{DR}}} = (\sim^*_{\Phi_{\mathsf{DR}}}, \gamma_{\Phi_{\mathsf{DR}}})$ where $\gamma_{\Phi_{\mathsf{DR}}}([u]) = \Phi_{\mathsf{DR}}(u)$ for all $u \in \Sigma^*$. Let us define another monitor $\mathcal{M} = (\sim, \gamma)$ such that $u_1 \sim u_2$ iff either $u_1, u_2 \in P$ or $u_1, u_2 \notin P$ for every $u_1, u_2 \in \Sigma^*$; and $\gamma([u]) = 1$ if $u \in P$, and $\gamma([u]) = 0$ if $u \notin P$. Note that for every $w \in \Sigma^\omega$ we have $w \in P$ iff infinitely many prefixes of w belong to P, therefore $\mathcal{M}$ has no limit error. However, it yields a prompt error of λ since it immediately outputs 0 instead of discounting on finite traces. Hence, $\mathcal{M}$ is a $(\lambda, 0)$-monitor for Φ_{DR}.*

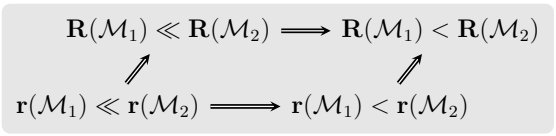


Figure 4.1: Implications between the comparisons of resource use.

Next, we prove that our definition constrains monitors not to make two equivalent traces too distant.

Proposition 4.2.5. *Let $\mathcal{M} = (\sim, \gamma)$ be a $(\delta_{\text{fin}}, \delta_{\lim})$-monitor for the property $\Phi = (\pi, \text{Val})$. For all $u_1, u_2 \in \Sigma^*$, if $u_1 \sim u_2$, then $|\Phi(u_1) - \Phi(u_2)| \leq 2\delta_{\text{fin}}$.*

Proof. By definition of $\mathcal{M}$ we have that $-\delta_{\text{fin}} \leq \pi(u_1) - \gamma([u_1]) \leq \delta_{\text{fin}}$ as well as $\delta_{\text{fin}} \geq -\pi(u_2) + \gamma([u_2]) \geq -\delta_{\text{fin}}$. If $u_1 \sim u_2$ then $\gamma([u_1]) = \gamma(us_2])$ and thus $-2\delta_{\text{fin}} \leq \pi(u_1) - \pi(u_2) \leq 2\delta_{\text{fin}}$. □

4.2.3 Resource Use of Abstract Monitors

As we demonstrated above, quantitative monitors may have different degrees of precision. A natural question is whether monitors with different error thresholds use a different amount of resources. To answer this question in its generality, we consider the following model-oblivious notions of resource use.

Definition 4.2.6 (Resource use)**.** *Let $\mathcal{M} = (\sim, \gamma)$ be a monitor. We consider two notions of resource use for $\mathcal{M}$ defined as functions from $\mathbb{N}$ to $\mathbb{N}$. We define* step-wise resource use *as $\mathbf{r}_n(\mathcal{M}) = |\Sigma^{\leq n}/\!\sim| - |\Sigma^{<n}/\!\sim|$ and* total resource use *as $\mathbf{R}_n(\mathcal{M}) = \sum_{i=0}^{n} \mathbf{r}_i(\mathcal{M}) = |\Sigma^{\leq n}/\!\sim|$.*

Given two monitors $\mathcal{M}_1$ and $\mathcal{M}_2$, we compare their resource use as follows. We write $\mathbf{r}(\mathcal{M}_1) < \mathbf{r}(\mathcal{M}_2)$ when there exists $n_0 \in \mathbb{N}$ such that for every $n \geq n_0$ we have $\mathbf{r}_n(\mathcal{M}_1) < \mathbf{r}_n(\mathcal{M}_2)$. In particular, when it holds for $n_0 = 1$, we write $\mathbf{r}(\mathcal{M}_1) \ll \mathbf{r}(\mathcal{M}_2)$. We define $\mathbf{R}(\mathcal{M}_1) < \mathbf{R}(\mathcal{M}_2)$ and $\mathbf{R}(\mathcal{M}_1) \ll \mathbf{R}(\mathcal{M}_2)$ similarly. Figure 4.1 shows how these notions relate. Moreover, definitions of $\mathbf{r}(\mathcal{M}_1) \propto \mathbf{r}(\mathcal{M}_2)$ and $\mathbf{R}(\mathcal{M}_1) \propto \mathbf{R}(\mathcal{M}_2)$ for $\propto \in \{\leq, \lll, >, \gg, \geq, \ggg\}$ are as expected.

The monitor $\mathcal{M}_1$ uses *at most as many* resources as $\mathcal{M}_2$ when we have $\mathbf{r}(\mathcal{M}_1) \lll \mathbf{r}(\mathcal{M}_2)$. If we further have $\mathbf{r}_n(\mathcal{M}_1) < \mathbf{r}_n(\mathcal{M}_2)$ for some $n \geq 1$, then $\mathcal{M}_1$ uses *fewer* resources than $\mathcal{M}_2$. We similarly define the cases for using *at least as many* and *more* resources.

Given a property Φ and a $(\delta_{\text{fin}}, \delta_{\lim})$-monitor $\mathcal{M}$ for Φ, we say that $\mathcal{M}$ is *resource optimal* for Φ when for every $(\delta_{\text{fin}}, \delta_{\lim})$-monitor $\mathcal{M}'$ for Φ we have $\mathbf{r}(\mathcal{M}) \lll \mathbf{r}(\mathcal{M}')$, i.e., $\mathcal{M}$ uses at most as many resources as any other monitor $\mathcal{M}'$ with the same error thresholds.

Example 4.2.7. *Recall from Examples 4.2.1 and 4.2.4 the discounted response property Φ_{DR}, its exact-value monitor $\mathcal{M}_{\Phi_{\text{DR}}}$, and the $(\lambda, 0)$-monitor $\mathcal{M}$. We claim that $\mathcal{M}$ uses fewer resources than $\mathcal{M}_{\Phi_{\text{DR}}}$. To show this, we first point out that $\mathbf{r}_0(\mathcal{M}) = \mathbf{r}_1(\mathcal{M}) = 1$ and $\mathbf{r}_n(\mathcal{M}) = 0$ for every $n \geq 2$. However, $\mathbf{r}_n(\mathcal{M}_{\Phi_{\text{DR}}}) \geq 1$ for every $n \geq 0$ because at each step the trace rq^n is not equivalent to any shorter trace. Therefore, while $\mathcal{M}_{\Phi_{\text{DR}}}$ is an infinite-state monitor, $\mathcal{M}$ is a finite-state monitor, and $\mathbf{r}(\mathcal{M}) < \mathbf{r}(\mathcal{M}_{\Phi_{\text{DR}}})$.*

Finally, we conclude the description of our framework by proving the implications in Figure 4.1 to establish how different ways to compare resource use of monitors relate as well as a refinement property for resource-optimal monitors.

Proposition 4.2.8. *For every monitor $\mathcal{M}_1$ and $\mathcal{M}_2$ the implications in Figure 4.1 hold.*

Proof. We only prove that $\mathbf{r}(\mathcal{M}_1) < \mathbf{r}(\mathcal{M}_2)$ implies $\mathbf{R}(\mathcal{M}_1) < \mathbf{R}(\mathcal{M}_2)$. The left-to-right implications in Figure 4.1 are trivial, and the other bottom-to-top implication follows from similar arguments. Assume by hypothesis that $\mathbf{r}(\mathcal{M}_1) < \mathbf{r}(\mathcal{M}_2)$, i.e., there exists $n_0 \in \mathbb{N}$ such that, $\mathbf{r}_n(\mathcal{M}_1) < \mathbf{r}_n(\mathcal{M}_2)$ for all $n \geq n_0$. Let $x_1 = \mathbf{R}_{n_0-1}(\mathcal{M}_1)$ and $x_2 = \mathbf{R}_{n_0-1}(\mathcal{M}_2)$. If $x_1 \leq x_2$, the implication holds because $\mathbf{r}_n(\mathcal{M}_1) < \mathbf{r}_n(\mathcal{M}_2)$ for all $n \geq n_0$ means that $\mathbf{R}_n(\mathcal{M}_1) < \mathbf{R}_n(\mathcal{M}_2)$ for all $n \geq n_0$. If $x_1 > x_2$, the following holds:

$$\begin{aligned}\mathbf{R}_{n_0+x_1-x_2}(\mathcal{M}_2) &= x_2 + \textstyle\sum_{i=n_0}^{n_0+x_1-x_2} \mathbf{r}_i(\mathcal{M}_2) \\ &\geq x_2 + \textstyle\sum_{i=n_0}^{n_0+x_1-x_2} (\mathbf{r}_i(\mathcal{M}_1) + 1) \\ &= x_2 + (x_1 - x_2 + 1) + \textstyle\sum_{i=n_0}^{n_0+x_1-x_2} \mathbf{r}_i(\mathcal{M}_1) \\ &> \mathbf{R}_{n_0+x_1-x_2}(\mathcal{M}_1)\end{aligned}$$

Hence $n_0' = n_0 + x_1 - x_2$ fulfills $\mathbf{R}_{n_0'}(\mathcal{M}_1) < \mathbf{R}_{n_0'}(\mathcal{M}_2)$. To conclude, for all $n \geq n_0'$ we have that

$$\mathbf{R}_n(\mathcal{M}_1) = \mathbf{R}_{n_0'}(\mathcal{M}_1) + \sum_{i=n_0'+1}^{n} \mathbf{r}_i(\mathcal{M}_1) < \mathbf{R}_{n_0'}(\mathcal{M}_2) + \sum_{i=n_0'+1}^{n} \mathbf{r}_i(\mathcal{M}_2) = \mathbf{R}_n(\mathcal{M}_2). \quad \square$$

Proposition 4.2.9. *Let Φ be a property and $\delta_{\mathsf{fin}}, \delta_{\mathrm{lim}}$ be two error thresholds. Consider two $(\delta_{\mathsf{fin}}, \delta_{\mathrm{lim}})$-monitors $\mathcal{M}_1 = (\sim_1, \gamma_1)$ and $\mathcal{M}_2 = (\sim_2, \gamma_2)$ for Φ. If $\sim_1 \subseteq \sim_2$ and $\mathcal{M}_1$ is resource optimal, then $\sim_1 = \sim_2$. Thus, $\mathcal{M}_2$ is also resource optimal.*

Proof. By resource optimality, we get $\mathbf{r}(\mathcal{M}_1) \lll \mathbf{r}(\mathcal{M}_2)$ which implies $|\Sigma^{\leq n}/{\sim_1}| \leq |\Sigma^{\leq n}/{\sim_2}|$ for all $n \in \mathbb{N}$, by Proposition 4.2.8. Hence $\sim_1$ has fewer equivalence classes compared to $\sim_2$. Finally, since $\sim_1 \subseteq \sim_2$, we must have that $\sim_1 = \sim_2$. $\square$

We remark that our definitional framework can be instantiated by existing monitor models, e.g., finite state automata [BLS11] or register monitors [FHS18, HS21]. More concretely, let us consider the discounted response property Φ_{DR} from Example 4.2.1. Its exact-value monitor $\mathcal{M}_{\Phi_{\mathsf{DR}}}$ from Example 4.2.4 can be implemented by a register monitor that stores the value n in its single register while checking for the LTL property P using its finite-state memory. On the other hand, the monitor $\mathcal{M}$ from Example 4.2.4 can be implemented by a finite state machine.

4.3 Approximate Prompt Monitoring

The original purpose of a monitor is to provide continuous feedback about the system status with respect to the property [BLS07, FFM12]. Focusing only on limit monitoring may allow an unbounded prompt error and thus fail to fulfill this task. In this section, we consider *prompt monitoring*, i.e., the case where the monitor performs bounded prompt error. First, we remark that considering a bounded prompt error implicitly bounds the limit error by definition.

Fact 4.3.1. *Let Φ be a property and $\delta_{\mathsf{fin}}, \delta_{\mathrm{lim}} \in \overline{\mathbb{R}}$ be error thresholds. If $\mathcal{M}$ is a $(\delta_{\mathsf{fin}}, \delta_{\mathrm{lim}})$-monitor for Φ, then it is also a $(\delta_{\mathsf{fin}}, x)$-monitor for Φ where $x = \min\{\delta_{\mathsf{fin}}, \delta_{\mathrm{lim}}\}$.*

$\Phi = (\pi, \mathsf{Lim})$ where:

$$\pi\colon u \mapsto \begin{cases} 0 & \text{if } u = \varepsilon \\ 3x & \text{if } u = a \\ 5x & \text{if } u = b \\ 7x & \text{if } u = c \\ 10x & \text{if } u \in \Sigma a \Sigma^* \\ 10x + y & \text{if } u \in \Sigma b \Sigma^* \\ 10x + 2y & \text{if } u \in \Sigma c \Sigma^* \end{cases}$$

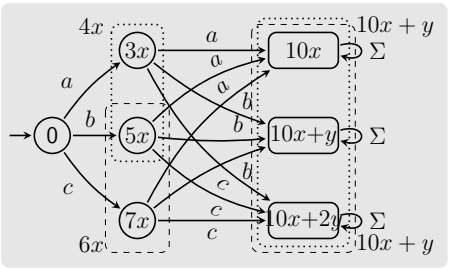


Figure 4.2: A property Φ over $\Sigma = \{a, b, c\}$ where $x > 0$ and $y \leq x$, and two resource-optimal (x, y)-monitors for Φ shown on top of the exact-value monitor $\mathcal{M}_\Phi$. As indicated by the output values on the dotted and dashed rectangles, the approximate monitors merge some equivalence classes of $\mathcal{M}_\Phi$ to save resources at the cost of losing precision.

4.3.1 Uniqueness of Resource-optimal Prompt Monitors

The exact-value monitor is arguably the most natural monitor for a given property. In fact, it is the unique error-free monitor that is resource optimal.

Theorem 4.3.2. *Let Φ be a property, and $\delta \in \overline{\mathbb{R}}$ be an error threshold. Then, $\mathcal{M}_\Phi$ is the unique resource-optimal $(0, \delta)$-monitor for Φ.*

Proof. Let $\Phi = (\pi, \mathsf{Val})$. Consider a resource-optimal $(0, \delta)$-monitor $\mathcal{M} = (\sim, \gamma)$ for Φ. We get $\sim \subseteq \sim_\Phi^*$ thanks to the following implications.

$$\begin{aligned} u_1 \sim u_2 &\implies \forall t \in \Sigma^* : u_1 t \sim u_2 t && \text{(right monotonicity)} \\ &\implies \forall t \in \Sigma^* : \gamma([u_1 t]) = \gamma([u_2 t]) && \text{(definition)} \\ &\implies \forall t \in \Sigma^* : \pi(u_1 t) = \pi(u_2 t) && \text{(prompt error 0)} \\ &\implies u_1 \sim_\Phi^* u_2 && \text{(definition)} \end{aligned}$$

On the one hand, we have that $\sim = \sim_\Phi^*$ by Proposition 4.2.9. On the other hand, we have that $\gamma([u]) = \pi(u)$ for all $u \in \Sigma^*$ since the prompt-error threshold is 0. As a direct consequence, $\mathcal{M} = \mathcal{M}_\Phi$. □

Unfortunately, the uniqueness of resource-optimal monitors does not necessarily hold once we allow erroneous monitor verdicts. For instance, Figure 4.2 shows on the left a property Φ parameterized by x and y, together with its exact-value monitor $\mathcal{M}_\Phi$ on the right. In addition, the figure highlights two ways to make $\sim_\Phi$ coarser to obtain distinct resource-optimal (x, y)-monitors for Φ.

Proposition 4.3.3. *For all $x > 0$ and $y \leq x$ there exists a property Φ that admits multiple resource-optimal (x, y)-monitors.*

4.3.2 Structure of Resource-optimal Prompt Monitors

Regardless of the uniqueness, one can ask whether making $\sim_\Phi$ coarser always yields a resource-optimal approximate monitor. Here, we answer this question negatively. In particular, Figure 4.3 shows on the left a property Φ and on the right a resource-optimal $(1, 0)$-monitor $\mathcal{M} = (\sim, \gamma)$ for Φ with $ab \nsim ba$, although $ab \sim_\Phi^* ba$.

$\Phi = (\pi, \mathsf{Lim})$ where:

$$\pi\colon u \mapsto \begin{cases} 0 & \text{if } u = \varepsilon \\ 3 & \text{if } u = c \\ 6 & \text{if } u = a \text{ or } u = ca \\ 9 & \text{if } u = b \text{ or } u = cb \\ 12 & \text{if } u = cab \\ 14 & \text{if } u = ab \text{ or } u = ba \\ 16 & \text{if } u = cba \\ 19 & \text{otherwise} \end{cases}$$

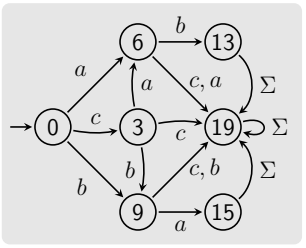


Figure 4.3: A property Φ for which no $(1,0)$-monitor that $\mathcal{M}_\Phi$ refines is resource optimal, and the witnessing resource-optimal approximate monitor that splits an equivalence class of the property.

Proposition 4.3.4. *There exist a property Φ and a $(1,0)$-monitor $\mathcal{M} = (\sim, \gamma)$ for Φ such that for every other $(1,0)$-monitor $\mathcal{M}' = (\sim', \gamma')$ we have that $\sim_\Phi \subseteq \sim'$ implies $\mathbf{r}(\mathcal{M}) \ll \mathbf{r}(\mathcal{M}')$.*

We established that the structure of the exact-value monitor does not necessarily provide insights into finding a resource-optimal approximate monitor. In fact, as we demonstrate in Figure 4.4, there exist a property such that its resource-optimal $(1,1)$-monitor $\mathcal{M}$ never minimizes the resource use $\mathbf{r}_i(\mathcal{M})$.

Proposition 4.3.5. *There exists a property Φ admitting a $(1,1)$-monitor $\mathcal{M} = (\sim, \gamma)$ such that for all right-monotonic equivalence relations $\approx$ over Σ^* and all $n \in \mathbb{N}$ we have*

$$|\Sigma^{\leq n}/{\sim}| > \min\left\{ \; |\Sigma^{\leq n}/{\approx}| \;\; \middle| \;\; \forall u_1, u_2 \in \Sigma^{\leq n} : u_1 \approx u_2 \Rightarrow |\Phi(u_1) - \Phi(u_2)| \leq 1 \; \right\}.$$

Proof. Let $\Phi = (\pi, \mathsf{LimSup})$ be a property from $\Sigma = \{a, b\}$ to $\overline{\mathbb{N}}$ where π is defined as follows.

$$\pi\colon u \mapsto \begin{cases} 8|u| & \text{if } u \in b^* \\ 8|u| - 16k + 4 & \text{if } u \in (b^+a^+)^k \text{ for some } k \geq 1 \\ 8|u| - 16k + 2 & \text{if } u \in (b^+a^+)^k b^+ \text{ for some } k \geq 1 \\ 8|u| - 2 & \text{if } u \in a^+ \\ 8|u| - 16k + 10 & \text{if } u \in (a^+b^+)^k \text{ for some } k \geq 1 \\ 8|u| - 16k - 4 & \text{if } u \in (a^+b^+)^k a^+ \text{ for some } k \geq 1 \end{cases}$$

Let $n \in \mathbb{N}$. The key argument is that it is beneficial to put a^n and b^n in the same equivalence class for minimizing $\mathbf{r}_n$ since $|\Phi(a^n) - \Phi(b^n)| = 2$ and since no other trace in $\Sigma^{\leq n}$ admits a value close to either $\Phi(a^n)$ or $\Phi(b^n)$. However, once we consider traces of length $n+1$, we introduce several values close to $\Phi(a^n)$ as well as $\Phi(b^n)$, but not both at the same time. Therefore, to minimize the resource use $\mathbf{r}_{n+1}$ while maintaining the prompt-error bound of 1, it becomes beneficial to put a^n and b^n in distinct equivalence classes. □

4.3.3 Unbounded Precision-resource Tradeoffs for Prompt Monitors

In this subsection, we exhibit properties admitting an infinite sequence of monitors that trade precision and resource use. First, we investigate the *maximal response-time* property by demonstrating how a monitor can save more and more resources by increasing both its prompt and limit errors.

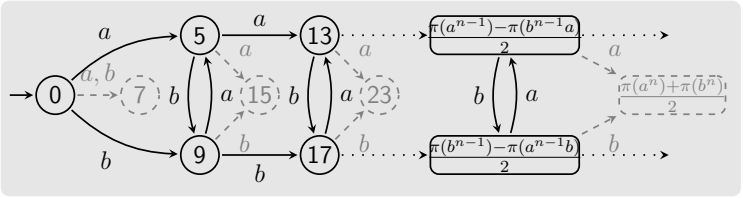


Figure 4.4: A resource-optimal (1,1)-monitor for the property Φ of Proposition 4.3.5 that never minimizes its step-wise resource use $\mathbf{r}_n$ (black). Attempting to minimize $\mathbf{r}_n$ at each step n results in taking a^n and b^n as equivalent, but breaking the equivalence at step $n+1$ as the prompt-error bound would be violated otherwise (gray).

Example 4.3.6. *Let $\Sigma = \{rq, ack, oo\}$ and consider the usual LTL response property $P = \square(rq \to \Diamond ack)$. We define* CurResp$(u) = 0$ *if $u \in P$, and* CurResp$(u) = |u| - |t|$ *otherwise, where $t \prec u$ is the longest prefix with $t \in P$. Now, let* MaxResp$(u) = \max_{t \preceq u}$ CurResp(t) *and define $\Phi_{\text{MR}} =$ (MaxResp, Lim), which we call the* maximal response-time *property. Note that* CurResp *outputs the current response time for a finite trace, and* MaxResp *outputs the maximum response time so far.*

Consider the monitor $\mathcal{M} = (\sim, \gamma)$ that counts the response time when there is an open rq, but only stores an approximation of the maximum when an ack occurs. More explicitly, let $\sim$ and γ be such that we have the following: $\mathcal{M}(u) = 5k+2$ if $u \in P$, where $k \in \mathbb{N}$ satisfies $5k \leq$ MaxResp$(u) < 5(k+1)$; and $\mathcal{M}(u) = \max\{\mathcal{M}(t),$ CurResp(u)$\}$ otherwise, where $t \prec u$ is the longest prefix with $t \in P$. We claim that $\mathcal{M}$ is a $(2,2)$-monitor for Φ_{MR}. First, observe that whenever there is no pending request, i.e., $u \in P$, the monitor has a prompt error of at most 2 by construction. Indeed, MaxResp$(u) \in \{5k+i \mid i \in \{0,1,2,3,4\}\}$. *In the case of a pending request, i.e., $u \notin P$, there is a prompt error only when the monitor's approximation of the maximum-so-far is not replaced by the current response time. Again, by construction, we can bound this error by 2. Intuitively, $\mathcal{M}$ achieves this approximation by merging in $\sim$ some equivalence classes of $\sim^*_{\Phi_{\text{MR}}}$ where there are no pending requests. One can thus verify that $\mathbf{r}(\mathcal{M}) < \mathbf{r}(\mathcal{M}_{\Phi_{\text{MR}}})$.*

The construction described in Example 4.3.6 can be generalized to identify a precision-resource tradeoff with an infinite hierarchy of approximate monitors.

Theorem 4.3.7. *For all $\delta \in \mathbb{N}$, there exists a (δ,δ)-monitor $\mathcal{M}_\delta$ for the maximal response-time property. Furthermore, $\mathbf{r}(\mathcal{M}_i) < \mathbf{r}(\mathcal{M}_j)$ for all $i > j$, and $\mathcal{M}_0$ is the exact-value monitor.*

Proof. Let $\Phi_{\text{MR}} =$ (MaxResp, Lim) be the maximal response-time property as introduced in Example 4.3.6. Let $\delta \in \mathbb{N}$ and $u \in \Sigma^*$. If u does not have a pending request, we define $\mathcal{M}_\delta(u) = k(2\delta+1)+\delta$ where $k \in \mathbb{N}$ satisfies $k(2\delta+1) \leq$ MaxResp$(u) < (k+1)(2\delta+1)$. Otherwise, if u has a pending request, we define $\mathcal{M}_\delta(u) = \max\{\mathcal{M}_\delta(t),$ CurResp(u)$\}$ where $t \prec u$ is the longest prefix with no pending request. We construct the (δ,δ)-monitor $\mathcal{M}_\delta$ for Φ_{MR} as in Example 4.3.6. In particular, $\mathcal{M}_0$ is the exact-value monitor. Indeed, $\delta = 0$ implies $\mathcal{M}_\delta(u) = k =$ MaxResp(u) when u does not have a pending request, and otherwise $\mathcal{M}_\delta(u) = \max_{t \preceq u}$ CurResp(t) $=$ MaxResp(u) by definition. For all $i > j$, the monitor $\mathcal{M}_i$ partitions the traces with no pending requests into sets of cardinality $2i+1$ while $\mathcal{M}_j$ does so using sets of cardinality $2j+1$. Then, the equivalence relation used by $\mathcal{M}_i$ is coarser than that of $\mathcal{M}_j$, and thus $\mathbf{r}(\mathcal{M}_i) < \mathbf{r}(\mathcal{M}_j)$. □

Note that, except $\mathcal{M}_0$, the monitors given by Theorem 4.3.7 have non-zero limit error. We explore in Section 4.4 the properties for which having fewer resources than the exact-value monitor forces a non-zero limit error. Moreover, we show in Example 4.4.6 that the maximal response time is one of these properties.

Next, we investigate the *server/client* property by demonstrating how a monitor can be more and more precise by increasing its resource use.

Example 4.3.8. *Consider a server that receives requests and issues acknowledgments. The number of simultaneous requests the system can handle is determined at runtime through a preprocessing computation. We describe a property that, at its core, requires that every request is acknowledged and the server never has more open requests than it can handle. In particular, until the server is turned off, the property assigns a value to each finite trace, denoting the likelihood and criticality of a potential immediate violation.*

Let $\Sigma = \{\mathtt{rq}, \mathtt{ack}, \mathtt{oo}, \mathtt{off}\}$ *be an alphabet,* $\lambda \in (0,1)$ *be a discount factor, and* $\Lambda > 0$ *be an integer denoting the request threshold. For every* $u \in \Sigma^*$ *we denote by* $\mathsf{NumReq}(u)$ *the number of pending requests in* u*. We define the* server/client property $\Phi_{\mathsf{SC}} = (\pi, \mathsf{Lim})$ *where* π *is defined as follows.*

- $\pi(u) = 0$ *if* u *contains an occurrence of* $\mathtt{off}$*,*
- $\pi(s) = \mathsf{NumReq}(u)\lambda^{|u|}$ *if* $\mathsf{NumReq}(t) \leq \Lambda$ *for all* $t \preceq u$*, and*
- $\pi(u) = \mathsf{NumReq}(t)\lambda^{|t|}$ *otherwise, where* $t \preceq u$ *is the shortest with* $\mathsf{NumReq}(t) > \Lambda$*.*

Theorem 4.3.9. *For every positive integer* Λ *and real number* $0 < \delta \leq \Lambda$*, there exists a* (δ, δ)*-monitor* $\mathcal{M}_\delta$ *for the server/client property* Φ_{SC}*. Furthermore,* $\mathcal{M}_\delta$ *uses finitely many resources.*

Proof. Let Λ and δ be as above, and consider the set X we define as follows: $X = \{u \in \Sigma^* \mid \sup_{t_1 \in \Sigma^*} \pi(ut_1) - \inf_{t_2 \in \Sigma^*} \pi(ut_2) \geq \delta\}$. We argue that X is finite. First, only a finite number of prefixes of a trace admitting an occurrence of $\mathtt{off}$ can belong to X since $\delta > 0$ and by definition of Φ_{SC}. Second, only a finite number of prefixes of a trace in which no $\mathtt{off}$ occurs can belong to X since the discounting forces the value of Φ_{SC} to converge to 0. We construct $\mathcal{M}_\delta$ such that, if the trace belongs to X, it outputs the value given by the property, otherwise it outputs the value of the shortest prefix that does not belong to X. In other words, $\mathcal{M}_\delta$ does not distinguish traces with the same prefix not belonging to X and thus admits at most $|\Sigma| \times |X|$ equivalence classes. □

4.4 Approximate Limit Monitoring

In contrast to Section 4.3 where we tackle the limit monitoring problem indirectly with a bounded prompt error, here we bound the limit error directly and allow arbitrary prompt error.

Example 4.4.1. *Let* $\Phi = (\pi, \mathsf{LimInf})$ *be a property over* $\Sigma = \{\mathtt{safe}, \mathtt{danger}, \mathtt{off}\}$ *such that* $\pi(u) = 2^{|t|}$ *if* u *does not contain* $\mathtt{off}$*, where* t *is the longest suffix of* u *of the form* $\mathtt{safe}^*$*, and* $\pi(u) = |u|_{\mathtt{danger}}$ *otherwise. Intuitively,* Φ *assigns each trace a confidence value while the system is on and how many times the system was in danger otherwise. We describe an approximate monitor with unbounded prompt error and bounded but non-zero limit error.*

Let $\sim$ be a right-monotonic equivalence relation and γ an output function such that $\mathcal{M} = (\sim, \gamma)$ satisfies the following: $\mathcal{M}(u) = \infty$ when u has no `off` *and ends with* `safe`*, $\mathcal{M}(u) = 0$ when u has no* `off` *and ends with* `danger`*, and $\mathcal{M}(u) = 9k + 4$ otherwise, where $k \in \mathbb{N}$ satisfies $9k \leq |u|_{\texttt{danger}} < 9(k+1)$. Notice that the monitor partitions $\mathbb{N}$ into intervals and takes traces with a "close enough" number of* `danger`*'s equivalent as in Example 4.3.6. It is easy to see that $\mathcal{M}$ is a $(\infty, 4)$-monitor for Φ.*

At its core, the limit error threshold of a monitor is a theoretical guarantee since we cannot compute arbitrary value functions at runtime. Then, as a starting point, we insist that the monitor has zero limit error, which is a reasonable requirement given that we allow unbounded prompt error. In this case, the monitoring is still potentially approximate since we allow any error on finite traces. To talk about properties for which saving resources by allowing prompt error is not possible, we define the following notion.

Definition 4.4.2 (Resource-intensive limit behavior)**.** *A property Φ has* resource-intensive limit behavior *iff its exact-value monitor $\mathcal{M}_\Phi$ is a resource-optimal $(\delta, 0)$-monitor for any $\delta \geq 0$.*

First, we identify a sufficient condition for a property to be resource-intensive limit behavior. Then, we present *reversible properties* and show a subclass of them that satisfy our condition.

4.4.1 Properties with Resource-intensive Limit Behavior

Let $\Phi = (\pi, \mathsf{Val})$ be a property and recall the equivalence $\sim^*_\Phi$: for every $u_1, u_2 \in \Sigma^*$ we have $u_1 \sim^*_\Phi u_2$ iff $\pi(u_1 t) = \pi(u_2 t)$ holds for all $t \in \Sigma^*$. To investigate the limit behavior of a property, we define the following equivalence relation: for every $u_1, u_2 \in \Sigma^*$ we have $u_1 \sim^\omega_\Phi u_2$ iff $\mathsf{Val}(\pi(u_1 w)) = \mathsf{Val}(\pi(u_2 w))$ holds for all $w \in \Sigma^\omega$. Intuitively, traces with indistinguishable limit behavior are equivalent according to this relation. As a direct consequence of Fact 4.3.1, the following holds.

Fact 4.4.3. *For every property Φ, we have that $\sim^*_\Phi \subseteq \sim^\omega_\Phi$.*

However, the converse does not necessarily hold, as we demonstrate with Example 4.4.4 below. We will show later that, when it holds, the property has resource-intensive limit behavior.

Example 4.4.4. *Recall the discounted response property Φ_{DR} in Example 4.2.1, and that for all $u, t \in \Sigma^*$, we have $u \sim^*_{\Phi_{\mathsf{DR}}} t$ iff either (i) both traces have no pending* `rq` *or (ii) both have a* `rq` *pending for the same number of steps.*

Let $u, t \in \Sigma^*$. We claim $u \sim^\omega_{\Phi_{\mathsf{DR}}} t$ iff either both traces have a pending request or both do not. Indeed, if u has a pending request and t does not, then we have $\Phi(u.\texttt{oo}^\omega) = 0$ but $\Phi(t.\texttt{oo}^\omega) = 1$. For the other direction, simply observe that if $u \sim^\omega_{\Phi_{\mathsf{DR}}} t$ then $\Phi(u.\texttt{oo}^\omega) = \Phi(t.\texttt{oo}^\omega)$, but the equality does not hold if u has a pending request and t does not (or vice versa). Having these characterizations at hand, we immediately observe that $u \sim^*_{\Phi_{\mathsf{DR}}} t$ implies $u \sim^\omega_{\Phi_{\mathsf{DR}}} t$.

Notice that the approximate monitor $\mathcal{M}$ for Φ_{DR} we constructed in Example 4.2.4 follows exactly the limit behavior of the property. We were able to take advantage of the fact that $\sim^\omega_{\Phi_{\mathsf{DR}}}$ is coarser than $\sim^*_{\Phi_{\mathsf{DR}}}$ and design $\mathcal{M}$ such that it saves resources by allowing some prompt error but no limit error. We generalize this observation by showing that we could not have designed such a monitor if these equivalences had overlapped.

Theorem 4.4.5. *Let $\Phi = (\pi, \mathsf{Val})$ be a property. If $\sim_\Phi^* = \sim_\Phi^\omega$ then Φ has resource-intensive limit behavior.*

Proof. Let $\mathcal{M} = (\sim, \gamma)$ be a resource-optimal $(\delta, 0)$-monitor for Φ. Suppose towards contradiction that $\sim_\Phi^* = \sim_\Phi^\omega$ and $\mathcal{M}_\Phi$ is not resource optimal for Φ. In particular $\sim \neq \sim_\Phi^*$. Since the limit-error threshold is 0, we get $\sim \subseteq \sim_\Phi^*$ by the following.

$$
\begin{aligned}
u_1 \sim u_2 &\implies \forall w \in \Sigma^\omega : \mathsf{Val}(\gamma([u_1 w])) = \mathsf{Val}(\gamma([u_2 w])) && \text{(right-monotonicity)}\\
&\iff \forall w \in \Sigma^\omega : \mathsf{Val}(\pi(u_1 w)) = \mathsf{Val}(\pi(u_2 w)) && \text{(limit error 0)}\\
&\iff u_1 \sim_\Phi^\omega u_2 && \text{(definition)}\\
&\iff u_1 \sim_\Phi^* u_2 && \text{(hypothesis)}
\end{aligned}
$$

The contradiction is then raised by Proposition 4.2.9 implying that $\sim = \sim_\Phi^*$. □

As demonstrated in Example 4.2.4 and discussed above, the discounted response property does not display resource-intensive limit behavior. We give below two examples of properties with resource-intensive limit behavior. Let us start with the *maximal response-time* property.

Example 4.4.6. *Consider the maximal response-time property $\Phi_{\mathsf{MR}} = (\mathsf{MaxResp}, \mathsf{Lim})$ from Example 4.3.6. We argue that $\sim_{\Phi_{\mathsf{MR}}}^*$ and $\sim_{\Phi_{\mathsf{MR}}}^\omega$ overlap.*

Suppose towards contradiction that there exist $u, t \in \Sigma^*$ such that $u \sim_{\Phi_{\mathsf{MR}}}^\omega t$ and $u \not\sim_{\Phi_{\mathsf{MR}}}^* t$. Then, there is $r \in \Sigma^*$ with $\Phi_{\mathsf{MR}}(ur) \neq \Phi_{\mathsf{MR}}(tr)$. If at least one of ur or tr has no pending request, take the continuation oo^ω to reach a contradiction to $u \sim_{\Phi_{\mathsf{MR}}}^\omega t$. Otherwise, if in both ur and tr the current response time is smaller than the maximum among granted requests, then the continuation ack^ω yields a contradiction. The same continuation covers the case when both current response times are greater. Finally, assume w.l.o.g. that the current response time is smaller than the maximum among granted requests in ur and greater in tr. In this case, ack^ω yields a contradiction again because their outputs stay the same as $\Phi_{\mathsf{MR}}(ur)$ and $\Phi_{\mathsf{MR}}(tr)$, respectively. Therefore, we have $u \sim_{\Phi_{\mathsf{MR}}}^* t$, and thus $\sim_{\Phi_{\mathsf{MR}}}^*$ and $\sim_{\Phi_{\mathsf{MR}}}^\omega$ overlap.

Next, we describe the *average response-time* property and argue that it displays resource-intensive limit behavior.

Example 4.4.7. *Let $\Sigma = \{\mathsf{rq}, \mathsf{ack}, \mathsf{oo}\}$ and consider the usual LTL response property $P = \square(\mathsf{rq} \to \Diamond \mathsf{ack})$. For $u \in \Sigma^*$, we denote by $\mathsf{RespTime}(u)$ the total number of letters between the matching rq-ack pairs in u, and by $\mathsf{NumReq}(u)$ the number of valid rq's in u, i.e., those that occur after the preceding request is acknowledged. Formally, we define them as follows. For all $u \in \Sigma^*$, we fix $f(u) = 1$ if $u \in P$, and $f(u) = 0$ otherwise. Then, we define $\mathsf{RespTime}(u) = \sum_{t \preceq u} 1 - f(r)$ and $\mathsf{NumReq}(u) = |P_u|$ where $P_u = \{t \preceq u \mid \exists r \in \Sigma^* : t = r.\mathsf{rq} \wedge p(t) = 1\}$ is the set of valid requests in u. We define the* average response-time *property as $\Phi_{\mathsf{AR}} = (\mathsf{AvgResp}, \liminf)$ where we let $\mathsf{AvgResp}(u) = \frac{\mathsf{RespTime}(u)}{\mathsf{NumReq}(u)}$ for all $u \in \Sigma^*$.*

We claim that $\sim_{\Phi_{\mathsf{AR}}}^*$ and $\sim_{\Phi_{\mathsf{AR}}}^\omega$ overlap. To show this, one can proceed similarly as in Example 4.4.6. The cases with no pending requests are similar. When both traces have a pending request and their output values differ, extend both with ack^ω to get a contradiction.

4.4.2 Reversible Properties

The *reversible* subclass of properties enjoys the ability to move between computation steps forward and backward deterministically. Such properties received particular interest in the literature since they can be implemented on hardware without energy dissipation [Lan61, Tof80]. Since it imitates the property, the exact-value monitor of a reversible property can roll back its computation, if allowed, without needing additional memory. From an automata-theoretic perspective, reversibility can be seen as the automaton being both forward and backward deterministic. Algebraically, this is captured by the syntactic monoid being a group.

Definition 4.4.8 (Reversible property)**.** *A property Φ is* reversible *iff $(\Sigma^*/\sim_\Phi^*, \cdot, \varepsilon)$ is a group.*

First, we describe the *average ping* property—a variant of the average response-time property where a single `ping` event captures `rq` and `ack` events, and time proceeds through clock `tk` events. We then show that this property is reversible.

Example 4.4.9. *Let $\Sigma = \{\texttt{ping}, \texttt{tk}, \texttt{oo}\}$. Let* $\mathsf{ValidTick}(u) = |u|_{\texttt{tk}} - |t|_{\texttt{tk}}$ *where $t \preceq u$ is the longest prefix with no* `ping`*, and let* $\mathsf{NumPing}(u) = |u|_{\texttt{ping}}$*. The* average ping *property is defined as $\Phi_{\mathsf{AP}} = (\mathsf{AvgPing}, \mathsf{LimInf})$ where, for all $u \in \Sigma^*$, we let* $\mathsf{AvgPing}(u) = \frac{\mathsf{ValidTick}(u)}{\mathsf{NumPing}(u)}$ *if* $\mathsf{NumPing}(u) > 0$*; and* $\mathsf{AvgPing}(u) = -1$ *otherwise.*

We argue that this property is reversible. To see why, first observe for all $u, t \in \Sigma^*$ that we have $u \sim_{\Phi_{\mathsf{AP}}}^* t$ iff (i) $\mathsf{NumPing}(u) = \mathsf{NumPing}(t)$ *and (ii)* $\mathsf{ValidTick}(u) = \mathsf{ValidTick}(t)$*. We particularly show for every $u, t, r \in \Sigma^*$ that if $u \nsim_{\Phi_{\mathsf{AP}}}^* t$ then $ur \nsim_{\Phi_{\mathsf{AP}}}^* tr$, therefore $\sim_{\Phi_{\mathsf{AP}}}^*$ yields a group. Let $u, t \in \Sigma^*$ be such that $u \nsim_{\Phi_{\mathsf{AP}}}^* t$ and let $r \in \Sigma^*$ be arbitrary. Suppose the condition (i) above does not hold. Since the* NumPing *values increase monotonically with every* `ping`*, we get* $\mathsf{NumPing}(ur) - \mathsf{NumPing}(tr) = \mathsf{NumPing}(u) - \mathsf{NumPing}(t)$*, which is non-zero by supposition. If (ii) does not hold, it does not hold for ur and tr either by a similar reasoning. Hence we have $ur \nsim_{\Phi_{\mathsf{AP}}}^* tr$.*

Intuitively, we can backtrack the information on these functions: The value of NumPing *is decremented with each preceding* `ping`*, while* ValidTick *is decremented with each preceding* `tk` *until it hits 0. It means that $\sim_{\Phi_{\mathsf{AP}}}^*$ can be seen as an automaton that is both forward and backward deterministic.*

We identify below a well-behaved subclass of reversible properties with resource-intensive limit behavior.

Theorem 4.4.10. *Let Φ be a reversible property. If for every $u, t \in \Sigma^*$ with $u \sim_\Phi^\omega t$ there exists $r \in \Sigma^*$ with $ur \sim_\Phi^* tr$, then Φ has resource-intensive limit behavior.*

Proof. We show that the reversibility of Φ, together with the above assumption, implies $\sim_\Phi^* = \sim_\Phi^\omega$. Note that the inclusion $\sim_\Phi^* \subseteq \sim_\Phi^\omega$ always holds as stated by Fact 4.4.3. Assuming $(\Sigma^*/\sim_\Phi^*, \cdot, \varepsilon)$ is a group, we have $u_1 t \sim_\Phi^* u_2 t \Rightarrow u_1 \sim_\Phi^* u_2$ for all $u_1, u_2, t \in \Sigma^*$. The inclusion $\sim_\Phi^\omega \subseteq \sim_\Phi^*$ holds since having $u_1 \nsim_\Phi^* u_2$ implies for all $t \in \Sigma^*$ that $u_1 t \nsim_\Phi^* u_2 t$, which in turn implies $u_1 \nsim_\Phi^\omega u_2$ by our initial assumption. Finally, by Theorem 4.4.5, we obtain that Φ has resource-intensive limit behavior. □

Recall the average ping property from Example 4.4.9. It is reversible, as discussed earlier, and satisfies the condition in Theorem 4.4.10, therefore it has resource-intensive limit behavior. Finally, we present the *maximal ping*—a similarly simple variant of the maximal response-time

property. We demonstrate that this property is not reversible, although it has resource-intensive limit behavior.

Example 4.4.11. *Let* $\Sigma = \{\texttt{ping}, \texttt{oo}\}$ *and consider the boolean property* $P = \square\lozenge\texttt{ping}$*. Let* CurPing$(u)$ *and* MaxPing(u) *be defined similarly as for the maximal response-time property in Example 4.3.6. We fix* $\Phi_{\mathsf{MP}} = (\mathsf{MaxPing}, \mathsf{Lim})$ *which we call the* maximal ping *property. Consider* $u = \texttt{ping.oo}$ *and* $t = \texttt{ping.oo.oo}$*. While* $u \not\sim^*_{\Phi_{\mathsf{MP}}} t$*, we have* $ut \sim^*_{\Phi_{\mathsf{MP}}} tt$*, therefore* $\sim^*_{\Phi_{\mathsf{MP}}}$ *does not yield a group. Intuitively, this is because we cannot backtrack the information on the running maximum. However, similarly as for the maximal response-time property in Example 4.3.6, one can verify that* $\sim^*_{\Phi_{\mathsf{MP}}} = \sim^{\omega}_{\Phi_{\mathsf{MP}}}$*.*

Note that a notion of reversibility exists for abstract monitors as well: A monitor $\mathcal{M} = (\sim, \gamma)$ where $\sim$ yields a group enjoys reversibility. In particular, this ability allows the monitor to return to a previous computation step without using additional resources and thus consider a different trace suffix.

4.5 Conclusion

We formalize a framework that supports reasoning about precision-resource tradeoffs for the approximate and exact monitoring of quantitative properties. Unlike previous results, which analyze tradeoffs for specific machine models such as register monitors [FHS18, HS21], the framework presented in this paper studies for the first time an abstract notion of monitors, independent of the representation model, and separates the monitor errors on finite traces from those at the limit. These innovations allow us to design and study monitors that keep the focus on the resources needed for the approximate monitoring of quantitative properties with a given precision. We provide several examples of when approximate monitoring can save resources and investigate when it fails to achieve this goal.

An expected future work is to provide a procedure for constructing a *concrete* (exact or approximate) monitor from an abstract description. Monitors having finitely many equivalence classes can be naturally mapped to finite-state automata. For a monitor with infinitely many equivalence classes, the model must be an infinite-state transition system. Yet, there are different levels of infinite state space. It can be generated, for example, by a finite collection of registers [FHS18] or by a pushdown system [DLT13a]. Even when two abstract monitors are mapped to register automata with the same number of registers, they may differ in the type of operations used or the run-time needed per observation. It is also worth emphasizing that saving a single register may save infinitely many resources. Our current results do not provide such performance, so it is a natural future direction. To this end, we can consider alternative approaches to evaluate a monitor based on the number of violations of the error threshold.

Another direction is on the relevance of resources through time. Our notion of resource use covers the number of equivalence classes added at time n, but an assumption that the monitor can release resources would trigger more possibilities. We can extend our framework to *dynamic abstract monitors* in a way that is related to existing works on dynamic programming for model checking [WYGG08]. Intuitively, a dynamic abstract monitor keeps track of the equivalence classes that can be reused in the future and prunes all the others to reduce resource use.

CHAPTER 5

Safety and Liveness of Quantitative Properties and Automata

In this chapter, the following publications were re-used in full:

- Thomas A. Henzinger, Nicolas Mazzocchi, N. Ege Saraç. *Quantitative Safety and Liveness*. In Foundations of Software Science and Computation Structures - 26th International Conference, **FoSSaCS 2023**.
- Udi Boker, Thomas A. Henzinger, Nicolas Mazzocchi, N. Ege Saraç. *Safety and Liveness of Quantitative Automata*. In 34th International Conference on Concurrency Theory, **CONCUR 2023**.
- Udi Boker, Thomas A. Henzinger, Nicolas Mazzocchi, N. Ege Saraç. *Safety and Liveness of Quantitative Properties and Automata*. In Logical Methods in Computer Science, **LMCS Volume 21 Issue 2 (2025)**.

5.1 Introduction

Boolean safety and liveness. Safety and liveness are elementary concepts in the semantics of computation [Lam77]. They can be explained through the thought experiment of a *ghost monitor*—an imaginary device that watches an infinite computation trace (word) at runtime, one observation (letter) at a time, and always maintains the set of *possible prediction values* to reflect the satisfaction of a given property. Let Φ be a boolean property, meaning that Φ divides all infinite traces into those that satisfy Φ, and those that violate Φ. After any finite number of observations, `True` is a possible prediction value for Φ if the observations seen so far are consistent with an infinite trace that satisfies Φ, and `False` is a possible prediction value for Φ if the observations seen so far are consistent with an infinite trace that violates Φ. When `True` is no possible prediction value, the ghost monitor can reject the hypothesis that Φ is satisfied. The property Φ is *safe* if and only if the ghost monitor can always reject a violating hypothesis Φ after a finite number of observations. Orthogonally, the property Φ is *live* if and only if the ghost monitor can never reject a hypothesis Φ after a finite number of observations: for all infinite traces, after every finite number of observations, `True` remains a possible prediction value for Φ.

The safety-liveness classification of properties is fundamental in verification. In the natural topology on infinite traces—the "Cantor topology"—the safety properties are the closed sets, and the liveness properties are the dense sets [AS85]. For every property Φ, the location of Φ within the Borel hierarchy that is induced by the Cantor topology—the so-called "safety-progress hierarchy" [CMP93]—indicates the level of difficulty encountered when verifying Φ. On the first level, we find the safety and co-safety properties, the latter being the complements of safety properties, i.e., the properties whose falsehood (rather than truth) can always be rejected after a finite number of observations by the ghost monitor. More sophisticated verification techniques are needed for second-level properties, which are the countable boolean combinations of first-level properties—the so-called "response" and "persistence" properties [CMP93]. Moreover, the orthogonality of safety and liveness leads to the following celebrated fact: *every* property can be written as the intersection of a safety property and a liveness property [AS85]. This means that every property Φ can be decomposed into two parts: a safety part—which is amenable to simple verification techniques, such as invariants—and a liveness part—which requires heavier verification paradigms, such as ranking functions. Dually, there is always a disjunctive decomposition of Φ into co-safety and co-liveness.

Quantitative safety and liveness. So far, we have retold the well-known story of safety and liveness for *boolean* properties. A boolean property Φ is formalized mathematically as the *set* of infinite computation traces that satisfy Φ, or equivalently, the characteristic *function* that maps each infinite trace to a truth value. Quantitative generalizations of the boolean setting allow us to capture not only correctness properties, but also performance properties [HO13]. In this chapter we reveal the story of safety and liveness for such *quantitative* properties, which are functions from infinite traces to an arbitrary set $\mathbb{D}$ of *values*. In order to compare values, we equip the value domain $\mathbb{D}$ with a partial order $<$, and we require $(\mathbb{D}, <)$ to be a complete lattice. The membership problem [CDH10b] for an infinite trace w and a quantitative property Φ asks whether $\Phi(w) \geq v$ for a given threshold value $v \in \mathbb{D}$. Correspondingly, in our thought experiment, the ghost monitor attempts to reject hypotheses of the form $\Phi(w) \geq v$, which cannot be rejected as long as all observations seen so far are consistent with an infinite trace w with $\Phi(w) \geq v$. We will define Φ to be a *quantitative safety* property if and only if every wrong hypothesis of the form $\Phi(w) \geq v$ can always be rejected by the ghost monitor after a finite number of observations, and we will define Φ to be a *quantitative liveness* property if and only if some wrong hypothesis of the form $\Phi(w) \geq v$ can never be rejected by the ghost monitor after any finite number of observations. We note that in the quantitative case, after every finite number of observations, the set of possible prediction values for Φ maintained by the ghost monitor may be finite or infinite, and in the latter case, it may not contain a minimal or maximal element.

Examples. Suppose we have four observations: observation `rq` for "request a resource," `gr` for "grant the resource," `tk` for "clock tick," and `oo` for "other." The boolean property *Resp* requires that every occurrence of `rq` in an infinite trace is followed eventually by an occurrence of `gr`. The boolean property *NoDoubleReq* requires that no occurrence of `rq` is followed by another `rq` without some `gr` in between. The quantitative property *MinRespTime* maps every infinite trace to the largest number k such that there are at least k occurrences of `tk` between each `rq` and the closest subsequent `gr`. The quantitative property *MaxRespTime* maps every infinite trace to the smallest number k such that there are at most k occurrences of `tk` between each `rq` and the closest subsequent `gr`. The quantitative property *AvgRespTime* maps every infinite trace to the lower limit value $\liminf$ of the infinite sequence $(v_i)_{i \geq 1}$, where v_i is, for the

first i occurrences of `tk`, the average number of occurrences of `tk` between `rq` and the closest subsequent `gr`. Note that the values of *AvgRespTime* can be ∞ for some computations, including those for which the value of *Resp* is `True`. This highlights that boolean properties are not embedded in the limit behavior of quantitative properties.

The boolean property *Resp* is live because every finite observation sequence can be extended with an occurrence of `gr`. In fact, *Resp* is a second-level liveness property (namely, a response property), because it can be written as a countable intersection of co-safety properties. The boolean property *NoDoubleReq* is safe because if it is violated, it will be rejected by the ghost monitor after a finite number of observations, namely, as soon as the ghost monitor sees a `rq` followed by another occurrence of `rq` without an intervening `gr`. According to our quantitative generalization of safety, *MinRespTime* is a safety property. The ghost monitor always maintains the minimal number k of occurrences of `tk` between any past `rq` and the closest subsequent `gr` seen so far; the set of possible prediction values for *MinRespTime* is then $\{0, 1, \ldots, k\}$. Every hypothesis of the form "the *MinRespTime*-value is at least v" is rejected by the ghost monitor as soon as $k < v$; if such a hypothesis is violated, this will happen after some finite number of observations. Symmetrically, the quantitative property *MaxRespTime* is co-safe, because every wrong hypothesis of the form "the *MaxRespTime*-value is at most v" will be rejected by the ghost monitor as soon as the smallest possible prediction value for *MaxRespTime*, which is the maximal number of occurrences of `tk` between any past `rq` and the closest subsequent `gr` seen so far, goes above v. By contrast, the quantitative property *AvgRespTime* is both live and co-live because no hypothesis of the form "the *AvgRespTime*-value is at least v," nor of the form "the *AvgRespTime*-value is at most v," can ever be rejected by the ghost monitor after a finite number of observations. All nonnegative real numbers and ∞ always remain possible prediction values for *AvgRespTime*. Note that a ghost monitor that attempts to reject hypotheses of the form $\Phi(w) \geq v$ does not need to maintain the entire set of possible prediction values, but only the $\sup$ of the set of possible prediction values, and whether or not the $\sup$ is contained in the set. Dually, updating the $\inf$ (and whether it is contained) suffices to reject hypotheses of the form $\Phi(w) \leq v$.

Quantitative safety and liveness in automata. The notions of safety and liveness consider system properties in full generality: every set of system executions—even the uncomputable ones—can be seen through the lens of the safety-liveness dichotomy. To bring these notions more in line with practical requirements, their projections onto formalisms with desirable closure and decidability properties, such as ω-regular languages, have been studied thoroughly in the boolean setting. For example, [AS87] gives a construction for the safety closure of a Büchi automaton and shows that Büchi automata are closed under the safety-liveness decomposition. In turn, [KV01] describes an efficient model-checking algorithm for Büchi automata that define safety properties.

Similarly to how boolean automata (e.g., regular and ω-regular automata) define classes of boolean properties amenable to boolean verification, quantitative automata (e.g., limit-average and discounted-sum automata) define classes of quantitative properties amenable to quantitative verification. Quantitative automata generalize standard boolean automata with weighted transitions and a value function that accumulates an infinite sequence of rational-valued weights into a single real number, a generalization of acceptance conditions of ω-regular automata.

We study the projection of the quantitative safety-liveness dichotomy onto the properties definable by common quantitative automata. First, we show how certain attributes of

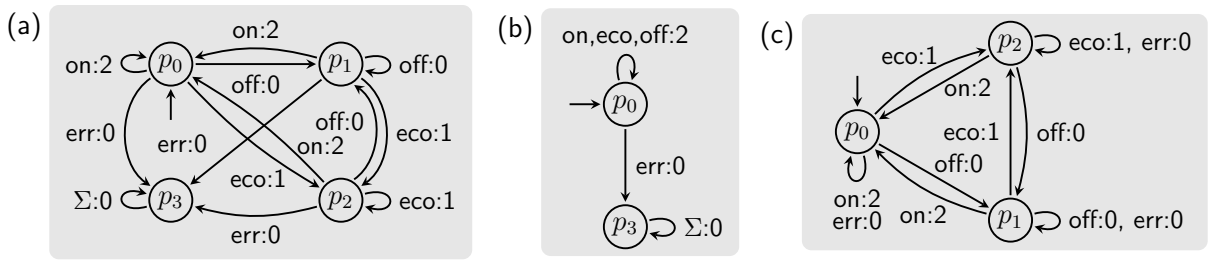


Figure 5.1: **(a)** A LimSup-automaton $\mathcal{A}$ modeling the long-term maximal power consumption of a device. **(b)** An Inf-automaton (or a LimSup-automaton) expressing the safety closure of $\mathcal{A}$. **(c)** A LimSup-automaton expressing the liveness component of the decomposition of $\mathcal{A}$.

quantitative automata simplify the notions of safety and liveness. Then, we use these simplifications to study safety and liveness of the classes of quantitative automata with the value functions Inf, Sup, LimInf, LimSup, LimInfAvg, LimSupAvg, and DSum [CDH10b]. In Figure 5.1a, we describe a quantitative automaton using the value function LimSup to express the long-term maximal power consumption of a device, which is neither safe nor live.

Contributions and overview. First, we focus on quantitative properties in their entire generality (Sections 5.2 to 5.6). We formally introduce quantitative safety as well as safety closure, namely the property that increases the value of each trace as little as possible to achieve safety. Then, we prove that our generalization of the boolean setting preserves classical desired behaviors. In particular, we show that a quantitative property Φ is safe if and only if Φ equals its safety closure. Moreover, for totally-ordered value domains, a quantitative property is safe if and only if for every value v, the set of executions whose value is at least v is safe in the boolean sense. We demonstrate a close relation between safety properties and continuous functions with respect to the dual Scott topology of their value domain. Pushing further, we define discounting properties on metrizable totally-ordered value domains, characterize them through uniform continuity, and show that they coincide with the conjunction of safety and co-safety.

We then generalize the safety-progress hierarchy to quantitative properties. We first define limit properties. For $\mathrm{Val} \in \{\mathsf{Inf}, \mathsf{Sup}, \mathsf{LimInf}, \mathsf{LimSup}\}$, the class of Val-properties captures those for which the value of each infinite trace can be derived by applying the limit function Val to the infinite sequence of values of finite prefixes. We prove that Inf-properties coincide with safety, Sup-properties with co-safety, LimInf-properties are suprema of countably many safety properties, and LimSup-properties infima of countably many co-safety properties. The LimInf-properties generalize the boolean persistence properties of [CMP93]; the LimSup-properties generalize their response properties. For example, *AvgRespTime* is a LimInf-property.

By defining quantitative safety and and co-safety via ghost monitors, we not only obtain a conservative and quantitative generalization of the boolean story, but also open up attractive frontiers for quantitative semantics, monitoring, and verification. For example, while the approximation of boolean properties reduces to adding and removing traces to and from a set, the approximation of quantitative properties offers a rich landscape of possibilities. In fact, we can approximate the notion of safety itself. Given an error bound α, the quantitative property Φ is *α-safe* if and only if for every value v and every infinite trace w whose value $\Phi(w)$ is less than v, all possible prediction values for Φ are less than $v + \alpha$ after some finite prefix of w. This means that, for an α-safe property Φ, the ghost monitor may not reject wrong hypotheses of the form $\Phi(w) \geq v$ after a finite number of observations, once the violation

is below the error bound. We show that every quantitative property that is both α-safe and β-co-safe, for any finite α and β, can be monitored arbitrarily precisely by a monitor that uses only a finite number of states.

We continue with introducing quantitative liveness and co-liveness, and prove that their relations with quantitative safety and co-safety further preserve the classical boolean facts. In particular, we show that in every value domain there is a unique property which is both safe and live, and then as a central result, we provide a safety-liveness decomposition that holds for every quantitative property, i.e., every quantitative property is the pointwise minimum of a safety and a liveness property. We also prove that, like for boolean properties, there exists a liveness-liveness decomposition for every nonunary quantitative property. Moreover, we provide alternative characterizations of liveness for quantitative properties that have the ability to express the least upper bound over their values, namely, supremum-closed. For such properties, we show that a property is live iff for every value v, the set of executions whose value is at least v is live in the boolean sense.

Second, we focus on quantitative automata (Sections 5.7 to 5.10). In contrast to general quantitative properties, these automata use functions on the totally-ordered domain of the real numbers (as opposed to a more general partially-ordered domain). Quantitative automata also have the restriction that only finitely many weights (those on the automaton transitions) can contribute to the value of an execution. In this setting, we carry the notion of safety (resp. co-safety, discounting) from properties to value functions, and show that a value function is safe (resp. co-safe, discounting) iff every quantitative automaton equipped with this value function expresses a safety (resp. co-safety, discounting) property. For example, Inf is a safe value function, and DSum is a discounting value function, therefore both safe and co-safe thanks to our characterization in the general setting.

We prove that the considered classes of quantitative automata are supremum-closed. Together with the total-order constraint, this helps us simplify the study of their safety and liveness thanks to our alternative characterizations from the first part. These simplified characterizations prove useful for checking safety and liveness of quantitative automata, constructing their safety closure, and decomposing them into safety and liveness components.

For example, let us recall the quantitative automaton in Figure 5.1a. Since it is supremum-closed, we can construct its safety closure in PTime by computing the maximal value it can achieve from each state. The safety closure of this automaton is shown in Figure 5.1b. For the value functions Inf, Sup, LimInf, LimSup, LimInfAvg, and LimSupAvg, the safety closure of a given automaton is an Inf-automaton, while for DSum, it is a DSum-automaton.

Evidently, one can check if a quantitative automaton $\mathcal{A}$ is safe by checking if it is equivalent to its safety closure, i.e., if $\mathcal{A}(w) = \mathit{SafetyCl}(\mathcal{A})(w)$ for every execution w. This allows for a PSpace procedure for checking the safety of Sup-, LimInf-, and LimSup-automata [CDH10b], but not for LimInfAvg- and LimSupAvg-automata, whose equivalence check is undecidable [DDG$^+$10, CDE$^+$10, HPPR18]. For these cases, we use the special structure of the safety-closure automaton for reducing safety checking to the problem of whether an automaton expresses a constant function. We show that the latter problem is PSpace-complete for LimInfAvg- and LimSupAvg-automata, by a somewhat involved reduction to the limitedness problem of distance automata, and obtain an $\mathrm{ExpSpace}$ decision procedure for their safety check.

Thanks to our alternative characterization of liveness, one can check if a quantitative automaton $\mathcal{A}$ is live by checking if its safety closure is universal with respect to its maximal value, i.e., if $\mathit{SafetyCl}(\mathcal{A})(w) \geq \top$ for every execution w, where $\top$ is the supremum over the values of $\mathcal{A}$.

For all value functions we consider except DSum, the safety closure is an Inf-automaton, which allows for a PSpace solution to liveness checking [KL07, CDH10b], which we show to be optimal. Yet, it is not applicable for DSum-automata, as the decidability of their universality check is an open problem. Nonetheless, as we consider only universality with respect to the maximal value of the automaton, we can reduce the problem again to checking whether an automaton expresses a constant function, which we show to be in PSpace for DSum-automata. This yields a PSpace-complete solution to the liveness check of DSum-automata.

Finally, we investigate the safety-liveness decomposition for quantitative automata. Recall the automaton from Figure 5.1a and its safety closure from Figure 5.1b. The liveness component of the corresponding decomposition is shown in Figure 5.1c. Intuitively, it ignores `err` and provides information on the power consumption as if the device never fails. Then, for every execution w, the value of the original automaton on w is the minimum of the values of its safety closure and the liveness component on w. Since we identified the value functions Inf and DSum as safe, their safety-liveness decomposition is trivial. For the classes of automata we study, we provide PTime safety-liveness decompositions. Moreover, for deterministic Sup-, LimInf-, and LimSup-automata, we give alternative PTime decompositions that preserve determinism.

We note that our alternative characterizations of safety and liveness of quantitative properties extend to co-safety and co-liveness. Our results for the specific automata classes are summarized in Table 5.1 and most are already implemented [CHMS24, CHMS25]. While we focus on automata that resolve nondeterminism by $\sup$, their duals hold for quantitative co-safety and co-liveness of automata that resolve nondeterminism by $\inf$, as well as for deterministic automata. We leave the questions of co-safety and co-liveness for automata that resolve nondeterminism by $\sup$ open.

Related work. To the best of our knowledge, previous definitions of safety and liveness in nonboolean domains make implicit assumptions about the specification language or implicitly use boolean safety and liveness [KSZ14, FK18, QSCP22, BV19]. We identify three notable exceptions – [WHK+13, LDL17, GS22].

In [WHK+13], the authors study a notion of safety for the rational-valued min-plus weighted automata on finite words. They take a weighted property as v-safe for a given rational v when for every execution w, if the hypothesis that the value of w is strictly less than v is wrong (i.e., its value is at least v), then there is a finite prefix of w to witness it. Then, a weighted property is safe when it is v-safe for *some* value v. Given a nondeterministic weighted automaton $\mathcal{A}$ and an integer v, they show that it is undecidable to check whether $\mathcal{A}$ is v-safe. By contrast, our definition quantifies over *all* values and nonstrict lower-bound hypotheses. Moreover, for this definition, we show that checking safety of all common classes of quantitative automata is decidable, even in the presence of nondeterminism.

In [LDL17], the authors present a safety-liveness decomposition on multi-valued truth domains, which are bounded distributive lattices. Their motivation is to provide algorithms for model-checking properties on multi-valued truth domains. While their definitions admit a safety-liveness decomposition, our definition of liveness captures strictly fewer properties, leading to a stronger safety-liveness decomposition theorem. In addition, our definitions also fit naturally with the definitions of emptiness, equivalence, and inclusion for quantitative languages [CDH10b].

In [GS22], the authors generalize the framework of [PH18] to nonboolean value domains. Their

	Inf	Sup, LimInf, LimSup	LimInfAvg, LimSupAvg	DSum
Safety closure construction	$O(1)$	PTIME Theorems 5.9.6 and 5.9.7		$O(1)$
Constant-function check	PSPACE-complete Theorem 5.8.2 and Theorems 5.8.3 and 5.8.8			
Safety check	$O(1)$	PSPACE-complete Theorem 5.9.10	EXPSPACE; PSPACE-hard Theorem 5.9.12 and Theorem 5.9.9	$O(1)$
Liveness check	PSPACE-complete Theorem 5.10.2			
Safety-liveness decomposition	$O(1)$	PTIME Theorems 5.10.3 to 5.10.5		$O(1)$

Table 5.1: The complexity of performing the operations on the left column with respect to nondeterministic automata with the value function specified on the top row.

definitions do not allow for a safety-liveness decomposition since their notion of safety is too permissive and their liveness too restrictive. They also do not have a fine-grained classification of nonsafety properties. We further elaborate on the relationships between the definitions of [LDL17, GS22] and ours in the relevant sections below.

Our study shows that determining whether a given quantitative automaton expresses a constant function is key to deciding safety and liveness, in particular for automata classes in which equivalence or universality checks are undecidable or open. To the best of our knowledge, this problem has not been studied before.

5.2 Quantitative Properties

Let $\Sigma = \{a, b, \ldots\}$ be a finite alphabet of letters (observations). An infinite (resp. finite) word (trace) is an infinite (resp. finite) sequence of letters $w \in \Sigma^\omega$ (resp. $u \in \Sigma^*$). For $n \in \mathbb{N}$, we denote by Σ^n the set of finite words of length n. Given $u \in \Sigma^*$ and $w \in \Sigma^* \cup \Sigma^\omega$, we write $u \prec w$ (resp. $u \preceq w$) when u is a strict (resp. nonstrict) prefix of w. We denote by $|w|$ the length of $w \in \Sigma^* \cup \Sigma^\omega$ and, given $a \in \Sigma$, by $|w|_a$ the number of occurrences of a in w. For $w \in \Sigma^* \cup \Sigma^\omega$ and $0 \leq i < |w|$, we denote by $w[i]$ the ith letter of w.

A *value domain* $\mathbb{D}$ is a poset. We assume that $\mathbb{D}$ is a nontrivial (i.e., $\bot \neq \top$) complete lattice. Whenever appropriate, we write 0 or $-\infty$ instead of $\bot$ for the least element $\inf \mathbb{D}$, and 1 or ∞ instead of $\top$ for the greatest element $\sup \mathbb{D}$. We respectively use the terms minimum and maximum for the greatest lower bound and the least upper bound of finitely many elements.

A *quantitative property* is a total function $\Phi : \Sigma^\omega \to \mathbb{D}$ from the set of infinite words to a value domain. A boolean property $P \subseteq \Sigma^\omega$ is a set of infinite words. We use the boolean domain $\mathbb{B} = \{0, 1\}$ with $0 < 1$ and, in place of P, its *characteristic property* $\Phi_P : \Sigma^\omega \to \mathbb{B}$, which is defined by $\Phi_P(w) = 1$ if $w \in P$, and $\Phi_P(w) = 0$ if $w \notin P$. When we say just *property*, we mean a quantitative one.

Given a property Φ and a finite word $u \in \Sigma^*$, let $P_{\Phi,u} = \{\Phi(uw) \mid w \in \Sigma^\omega\}$. A property Φ is $\sup$-*closed* (resp. $\inf$-*closed*) when for every finite word $u \in \Sigma^*$ we have that $\sup P_{\Phi,u} \in P_{\Phi,u}$ (resp. $\inf P_{\Phi,u} \in P_{\Phi,u}$).

Given a property $\Phi : \Sigma^\omega \to \mathbb{D}$ and a value $v \in \mathbb{D}$, we define $\Phi_{\sim v} = \{w \in \Sigma^\omega \mid \Phi(w) \sim v\}$ for $\sim \in \{\leq, \geq, \not\leq, \not\geq\}$. The *top value* of a property Φ is $\sup_{w \in \Sigma^\omega} \Phi(w)$, which we denote

by $\top_\Phi$. For all properties Φ_1, Φ_2 on a value domain $\mathbb{D}$ and all words $w \in \Sigma^\omega$, we let $\min(\Phi_1, \Phi_2)(w) = \min(\Phi_1(w), \Phi_2(w))$ and $\max(\Phi_1, \Phi_2)(w) = \max(\Phi_1(w), \Phi_2(w))$. For a value domain $\mathbb{D}$, the *inverse* of $\mathbb{D}$ is the domain $\overline{\mathbb{D}}$ that contains the same elements as $\mathbb{D}$ but with the ordering reversed. For a property Φ, we define its *complement* $\overline{\Phi} : \Sigma^\omega \to \overline{\mathbb{D}}$ by $\overline{\Phi}(w) = \Phi(w)$ for all $w \in \Sigma^\omega$.

Some properties can be defined as limits of value sequences. A *finitary property* $\pi \colon \Sigma^* \to \mathbb{D}$ associates a value with each finite word. A *value function* $\mathsf{Val} \colon \mathbb{D}^\omega \to \mathbb{D}$ condenses an infinite sequence of values to a single value. Given a finitary property π, a value function Val, and a word $w \in \Sigma^\omega$, we write $\mathsf{Val}_{u \prec w} \pi(u)$ instead of $\mathsf{Val}(\pi(u_0)\pi(u_1)\ldots)$, where each u_i satisfies $u_i \prec w$ and $|u_i| = i$.

5.3 Quantitative Safety

A boolean property $P \subseteq \Sigma^\omega$ is safe in the boolean sense iff for every $w \notin P$ there is a prefix $u \prec w$ with $uw' \notin P$ for all $w' \in \Sigma^\omega$ [AS85], in other words, every wrong membership hypothesis has a finite witness. Given a property $\Phi : \Sigma^\omega \to \mathbb{D}$, a trace $w \in \Sigma^\omega$, and a value $v \in \mathbb{D}$, the quantitative membership problem [CDH10b] asks whether $\Phi(w) \geq v$. We define quantitative safety as follows: the property Φ is safe iff every wrong hypothesis of the form $\Phi(w) \geq v$ has a finite witness $u \prec w$.

Definition 5.3.1 (Safety). *A property $\Phi : \Sigma^\omega \to \mathbb{D}$ is* safe *when for every $w \in \Sigma^\omega$ and value $v \in \mathbb{D}$ with $\Phi(w) \not\geq v$, there is a prefix $u \prec w$ such that $\sup_{w' \in \Sigma^\omega} \Phi(uw') \not\geq v$.*

Let us illustrate this definition with the *minimal response-time* property.

Example 5.3.2. *Let $\Sigma = \{\mathsf{rq}, \mathsf{gr}, \mathsf{tk}, \mathsf{oo}\}$ and $\mathbb{D} = \mathbb{N} \cup \{\infty\}$. We define the minimal response-time property $\Phi_{\min}$ through an auxiliary finitary property $\pi_{\min}$ that computes the minimum response time so far. In a finite or infinite trace, an occurrence of rq is* granted *if it is followed, later, by a gr, and otherwise it is* pending*. Let $\pi_{\mathsf{last}}(u) = \infty$ if the finite trace u contains a pending rq, or no rq, and $\pi_{\mathsf{last}}(u) = |u'|_{\mathsf{tk}} - |u''|_{\mathsf{tk}}$ otherwise, where $u' \prec u$ is the longest prefix of u with a pending rq, and $u'' \prec u'$ is the longest prefix of u' without pending rq. Intuitively, π_{last} provides the response time for the last request when all requests are granted, and ∞ when there is a pending request or no request. Given $u \in \Sigma^*$, taking the minimum of the values of π_{last} over the prefixes $u' \preceq u$ gives us the minimum response time so far. Let $\pi_{\min}(u) = \min_{u' \preceq u} \pi_{\mathsf{last}}(u')$ for all $u \in \Sigma^*$, and $\Phi_{\min}(w) = \lim_{u \prec w} \pi_{\min}(u)$ for all $w \in \Sigma^\omega$. The limit always exists because $\pi_{\min}$ is nonincreasing.*

The minimal response-time property is safe. Let $w \in \Sigma^\omega$ and $v \in \mathbb{D}$ such that $\Phi_{\min}(w) < v$. Then, some prefix $u \prec w$ contains a rq that is granted after $v' < v$ ticks, in which case, no matter what happens in the future, the minimal response time is guaranteed to be at most v'; that is, $\sup_{w' \in \Sigma^\omega} \Phi_{\min}(uw') \leq v' < v$. Recalling from the introduction the ghost monitor that maintains the $\sup$ of possible prediction values for the minimal response-time property, that value is always $\pi_{\min}$; that is, $\sup_{w' \in \Sigma^\omega} \Phi_{\min}(uw') = \pi_{\min}(u)$ for all $u \in \Sigma^*$. Note that in the case of minimal response time, the $\sup$ of possible prediction values is always realizable; that is, for all $u \in \Sigma^*$, there exists $w \in \Sigma^\omega$ such that $\sup_{w' \in \Sigma^\omega} \Phi_{\min}(uw') = \Phi_{\min}(uw)$.

We first show that our definition of safety generalizes the boolean one.

Proposition 5.3.3. *Quantitative safety generalizes boolean safety. In particular, for every boolean property $P \subseteq \Sigma^\omega$, the following statements are equivalent:*

1. *P is safe according to the classical definition [AS85].*
2. *The characteristic property Φ_P is safe.*
3. *For every $w \in \Sigma^\omega$ and $v \in \mathbb{B}$ with $\Phi_P(w) < v$, there exists a prefix $u \prec w$ such that for all $w' \in \Sigma^\omega$, we have $\Phi_P(uw') < v$.*

Proof. Recall that (1) means the following: for every $w \notin P$ there exists $u \prec w$ such that for all $w' \in \Sigma^\omega$ we have $uw' \notin P$. Expressing the same statement with the characteristic property Φ_P of P gives us for every $w \in \Sigma^\omega$ with $\Phi_P(w) = 0$ there exists $u \prec w$ such that for all $w' \in \Sigma^\omega$ we have $\Phi_P(uw') = 0$. In particular, since $\mathbb{B} = \{0, 1\}$ and $0 < 1$, we have for every $w \in \Sigma^\omega$ with $\Phi_P(w) < 1$ there exists $u \prec w$ such that for all $w' \in \Sigma^\omega$ we have $\Phi_P(uw') < 1$. Moreover, since there is no $w \in \Sigma^\omega$ with $\Phi_P(w) < 0$, we get the equivalence between (1) and (3). Now, observe that for every $u \in \Sigma^*$, we have $\Phi_P(uw') < 1$ for all $w' \in \Sigma^\omega$ iff $\sup_{w' \in \Sigma^\omega} \Phi_P(uw') < 1$, simply because the domain $\mathbb{B}$ is a finite total order. Therefore, (2) and (3) are equivalent as well. □

Next, we show that safety properties are closed under pairwise $\min$ and $\max$.

Proposition 5.3.4. *For every value domain $\mathbb{D}$, the set of safety properties over $\mathbb{D}$ is closed under $\min$ and $\max$.*

Proof. First, consider the two safety properties Φ_1, Φ_2 and let Φ be their pairwise minimum, i.e., $\Phi(w) = \min(\Phi_1(w), \Phi_2(w))$ for all $w \in \Sigma^\omega$. Suppose towards contradiction that Φ is not safe, i.e., for some $w \in \Sigma^\omega$ and $v \in \mathbb{D}$ such that $\Phi(w) \not\geq v$ and $\sup_{w' \in \Sigma^\omega} \Phi(uw') \geq v$ for all $u \prec w$. Observe that $\Phi(w) \not\geq v$ implies $\Phi_1(w) \not\geq v$ or $\Phi_2(w) \not\geq v$. We assume without loss of generality that $\Phi_1(w) \not\geq v$ holds. Thanks to the safety of Φ_1, there exists $u' \prec w$ such that $\sup_{w' \in \Sigma^\omega} \Phi_1(u'w') \not\geq v$. Since $\Phi_1(u'w') \geq \Phi(u'w')$ for all $w' \in \Sigma^\omega$, we have that $\sup_{w' \in \Sigma^\omega} \Phi_1(u'w') \geq \sup_{u'w' \in \Sigma^\omega} \Phi(u'w') \geq v$. This implies that $\sup_{w' \in \Sigma^\omega} \Phi_1(u'w') \geq v$, which yields a contradiction.

Now, consider the two safety properties Φ_1, Φ_2 and let Φ be their pairwise maximum, i.e., $\Phi(w) = \max(\Phi_1(w), \Phi_2(w))$ for all $w \in \Sigma^\omega$. Suppose towards contradiction that Φ is not safe, i.e., for some $w \in \Sigma^\omega$ and $v \in \mathbb{D}$, we have $\Phi(w) \not\geq v$ and $\sup_{w' \in \Sigma^\omega} \Phi(uw') \geq v$ for all $u \prec w$. Due to safety of both Φ_1 and Φ_2, we get for each $i \in \{1, 2\}$ the following: for all $w \in \Sigma^\omega$ and $v \in \mathbb{D}$ if $\Phi_i(w) \not\geq v$ there is $u_i \prec w$ such that $\sup_{w' \in \Sigma^\omega} \Phi(u_i w') \not\geq v$. Combining the two statements, we get for all $w \in \Sigma^\omega$ and $v \in \mathbb{D}$ if $\max(\Phi_1(w), \Phi_2(w)) \not\geq v$, then there exists $u \prec w$ such that $\max(\sup_{w' \in \Sigma^\omega} \Phi_1(uw'), \sup_{w' \in \Sigma^\omega} \Phi_2(uw')) \not\geq v$. In particular, $\max(\sup_{w' \in \Sigma^\omega} \Phi_1(uw'), \sup_{w' \in \Sigma^\omega} \Phi_2(uw')) \not\geq v$ holds since $\max(\Phi_1(w), \Phi_2(w)) = \Phi(w) \not\geq v$. Since $\sup(X \cup Y) = \max(\sup X, \sup Y)$ for all $X, Y \subseteq \mathbb{D}$, we get

$$\sup_{w' \in \Sigma^\omega} (\max(\Phi_1(uw'), \Phi_2(uw'))) = \max\left(\sup_{w' \in \Sigma^\omega} \Phi_1(uw'), \sup_{w' \in \Sigma^\omega} \Phi_2(uw') \right).$$

Consequently,

$$\sup_{w' \in \Sigma^\omega} \max(\Phi_1(uw'), \Phi_2(uw')) = \sup_{w' \in \Sigma^\omega} \Phi(uw') \not\geq v,$$

thus, a contradiction. □

We now generalize the notion of safety closure and present an operation that makes a property safe by increasing the value of each trace as little as possible.

Definition 5.3.5 (Safety closure)**.** *The* safety closure *of a property Φ is the property* $SafetyCl(\Phi)$ *defined by* $SafetyCl(\Phi)(w) = \inf_{u \prec w} \sup_{w' \in \Sigma^\omega} \Phi(uw')$ *for all* $w \in \Sigma^\omega$.

We can say the following about the safety closure operation.

Theorem 5.3.6. *For every property $\Phi : \Sigma^\omega \to \mathbb{D}$, the following statements hold.*

1. $SafetyCl(\Phi)$ *is safe.*
2. $SafetyCl(\Phi)(w) \geq \Phi(w)$ *for all* $w \in \Sigma^\omega$.
3. $SafetyCl(\Phi)(w) = SafetyCl(SafetyCl(\Phi))(w)$ *for all* $w \in \Sigma^\omega$.
4. Φ *is safe iff* $\Phi(w) = SafetyCl(\Phi)(w)$ *for all* $w \in \Sigma^\omega$.
5. *For every safety property* $\Psi : \Sigma^\omega \to \mathbb{D}$, *if* $\Phi(w) \leq \Psi(w)$ *for all* $w \in \Sigma^\omega$, *then* $SafetyCl(\Phi)(w) \leq \Psi(w)$ *for all* $w \in \Sigma^\omega$.

Proof. We first prove that $\sup_{w' \in \Sigma^\omega} SafetyCl(\Phi)(uw') \leq \sup_{w' \in \Sigma^\omega} \Phi(uw')$ for all $u \in \Sigma^*$, in other words, $\sup_{w' \in \Sigma^\omega} \inf_{u' \prec uw'} \sup_{w'' \in \Sigma^\omega} \Phi(u'w'') \leq \sup_{w' \in \Sigma^\omega} \Phi(uw')$ for all $u \in \Sigma^*$. This will be useful for the proofs of the first and the third items above.

$$
\begin{aligned}
&\forall u : \sup_{w' \in \Sigma^\omega} \Phi(uw') \in \{\sup_{w'' \in \Sigma^\omega} \Phi(u'w'') \mid u' \preceq u\} \\
&\quad \implies \forall u : \sup_{w' \in \Sigma^\omega} \Phi(uw') \geq \inf_{u' \preceq u} \sup_{w'' \in \Sigma^\omega} \Phi(u'w'') \\
&\quad \implies \forall u : \sup_{w' \in \Sigma^\omega} \Phi(uw') \geq \sup_{w' \in \Sigma^\omega} \inf_{u' \preceq u} \sup_{w'' \in \Sigma^\omega} \Phi(u'w'') \qquad (\dagger) \\
&\forall u, t : \sup_{w' \in \Sigma^\omega} \Phi(uw') \geq \sup_{w'' \in \Sigma^\omega} \Phi(utw'') \\
&\quad \implies \forall u : \sup_{w' \in \Sigma^\omega} \Phi(uw') \geq \sup_{w' \in \Sigma^\omega} \inf_{t \prec w'} \sup_{w'' \in \Sigma^\omega} \Phi(utw'') \qquad (\ddagger) \\
&(\dagger) \wedge (\ddagger) \implies \forall u : \sup_{w' \in \Sigma^\omega} \Phi(uw') \geq \sup_{w' \in \Sigma^\omega} \inf_{u' \prec uw'} \sup_{w'' \in \Sigma^\omega} \Phi(u'w'')
\end{aligned}
$$

1. Now, we prove that $SafetyCl(\Phi)$ is safe. Suppose $SafetyCl(\Phi)$ is not safe, i.e., there exist w and v for which $SafetyCl(\Phi)(w) \not\geq v$ and $\sup_{w' \in \Sigma^\omega} SafetyCl(\Phi)(uw') \geq v$ for all $u \prec w$. As a direct consequence of the fact that $\sup_{w' \in \Sigma^\omega} SafetyCl(\Phi)(uw') \leq \sup_{w' \in \Sigma^\omega} \Phi(uw')$ for all $u \in \Sigma^*$, we have that $\inf_{u \prec w} \sup_{w' \in \Sigma^\omega} \Phi(uw') \geq v$. It implies that $SafetyCl(\Phi)(w) \geq v$, which contradicts the hypothesis $SafetyCl(\Phi)(w) \not\geq v$. Hence $SafetyCl(\Phi)$ is safe.

2. Next, we prove that $SafetyCl(\Phi)(w) \geq \Phi(w)$ for all $w \in \Sigma^\omega$. Given $u \in \Sigma^*$, let $P_{\Phi,u} = \{\Phi(uw') \mid w' \in \Sigma^\omega\}$. Observe that $SafetyCl(\Phi)(w) = \lim_{u \prec w}(\sup P_{\Phi,u})$ for all $w \in \Sigma^\omega$. Moreover, $\Phi(w) \in P_{\Phi,u}$ for each $u \prec w$, and thus $\sup P_{\Phi,u} \geq \Phi(w)$ for each $u \prec w$, which implies $\lim_{u \prec w}(\sup P_{\Phi,u}) \geq \Phi(w)$, since the sequence of suprema is nonincreasing.

3. Next, we prove that $SafetyCl(\Phi)(w) = SafetyCl(SafetyCl(\Phi))(w)$ for all $w \in \Sigma^\omega$. Recall from the first paragraph that $\sup_{w' \in \Sigma^\omega} SafetyCl(\Phi)(uw') \leq \sup_{w' \in \Sigma^\omega} \Phi(uw')$ for all $u \in \Sigma^*$. So, for every $w \in \Sigma^\omega$, we have $\inf_{u \prec w} \sup_{w' \in \Sigma^\omega} SafetyCl(\Phi)(uw') \leq \inf_{u \prec w} \sup_{w' \in \Sigma^\omega} \Phi(uw')$ and thus $SafetyCl(SafetyCl(\Phi))(w) \leq SafetyCl(\Phi)(w)$ for all $w \in \Sigma^\omega$. Since we also have $SafetyCl(SafetyCl(\Phi))(w) \geq SafetyCl(\Phi)(w)$, then the equality holds for all $w \in \Sigma^\omega$.

4. Next, we prove that Φ is safe iff $\Phi(w) = \mathit{SafetyCl}(\Phi)(w)$ for all $w \in \Sigma^\omega$. The right-to-left implication follows from the fact that $\mathit{SafetyCl}(\Phi)$ is safe, as proved above in item (1). Now, assume Φ is safe, i.e., for all $w \in \Sigma^\omega$ and $v \in \mathbb{D}$ if $\Phi(w) \not\geq v$ then there exists $u \prec w$ with $\sup_{w' \in \Sigma^\omega} \Phi(uw') \not\geq v$. Suppose towards contradiction that for some $x \in \Sigma^\omega$ we have $\Phi(x) < \mathit{SafetyCl}(\Phi)(x) = \inf_{u \prec x} \sup_{w' \in \Sigma^\omega} \Phi(uw')$. Let $v = \inf_{u \prec x} \sup_{w' \in \Sigma^\omega} \Phi(uw')$. Since Φ is safe and $\Phi(x) \not\geq v$, there exists $u' \prec x$ such that $\sup_{w' \in \Sigma^\omega} \Phi(u'w') \not\geq v$. Observe that for all $x \in \Sigma^\omega$ and $u_1 \prec u_2 \prec x$ we have $\sup_{w' \in \Sigma^\omega} \Phi(u_2 w') \leq \sup_{w' \in \Sigma^\omega} \Phi(u_1 g)$, i.e., the supremum is nonincreasing with longer prefixes. Therefore, we have $\inf_{u \prec x} \sup_{w' \in \Sigma^\omega} \Phi(uw') \leq \sup_{w' \in \Sigma^\omega} \Phi(u'w')$. But since $\sup_{w' \in \Sigma^\omega} \Phi(u'w') \not\geq v$, we get a contradiction.

5. Finally, we prove that $\mathit{SafetyCl}(\Phi)$ is the least safety property that bounds Φ from above. Assume there exists a safety property Ψ such that $\Phi(w) \leq \Psi(w)$ holds for all $w \in \Sigma^\omega$. Then, for every infinite word $w \in \Sigma^\omega$ and all of its prefixes $u \prec w$ we have $\Phi(uw') \leq \Psi(uw')$ for all $w' \in \Sigma^\omega$. It implies for every $w \in \Sigma^\omega$ and every $u \prec w$, we have $\sup_{w' \in \Sigma^\omega} \Phi(uw') \leq \sup_{w' \in \Sigma^\omega} \Psi(uw')$. Then, for every $w \in \Sigma^\omega$, we have $\inf_{u \prec w} \sup_{w' \in \Sigma^\omega} \Phi(uw') \leq \inf_{u \prec w} \sup_{w' \in \Sigma^\omega} \Psi(uw')$. By definition, this is the same as $\mathit{SafetyCl}(\Phi)(w) \leq \mathit{SafetyCl}(\Psi)(w)$ for all $w \in \Sigma^\omega$. Moreover, since Ψ is safe, it is equivalent to its safety closure as we proved above, and thus $\mathit{SafetyCl}(\Phi)(w) \leq \Psi(w)$ for all $w \in \Sigma^\omega$. □

We note that a property's safety remains unaffected by the top value of its domain.

Remark 5.3.7. *Consider a property $\Phi : \Sigma^\omega \to \mathbb{D}$. If Φ is safe, it remains safe after removing (resp. adding) values greater than $\top_\Phi$ from $\mathbb{D}$ (resp. to $\mathbb{D}$). In particular, consider the value domains $\mathbb{D}_\Phi = \{v \in \mathbb{D} \mid v \leq \top_\Phi\}$ and $\mathbb{D}' = \mathbb{D} \cup \{\top'\}$ with $v < \top'$ for all $v \in \mathbb{D}$. It is easy to see that if Φ is safe, then $\Phi_1 : \Sigma^\omega \to \mathbb{D}_\Phi$ and $\Phi_2 : \Sigma^\omega \to \mathbb{D}'$ where $\Phi(w) = \Phi_1(w) = \Phi_2(w)$ for all $w \in \Sigma^\omega$ are also safe.*

Recall that a safety property allows rejecting wrong lower-bound hypotheses with a finite witness by assigning a tight upper bound to each trace. We define co-safety properties symmetrically: a property Φ is co-safe iff every wrong hypothesis of the form $\Phi(w) \leq v$ has a finite witness $u \prec w$.

Definition 5.3.8 (Co-safety)**.** *A property $\Phi : \Sigma^\omega \to \mathbb{D}$ is* co-safe *when for every $w \in \Sigma^\omega$ and value $v \in \mathbb{D}$ with $\Phi(w) \not\leq v$, there exists a prefix $u \prec w$ such that $\inf_{w' \in \Sigma^\omega} \Phi(uw') \not\leq v$.*

Definition 5.3.9 (Co-safety closure)**.** *The* co-safety closure *of a property Φ is the property $\mathit{CoSafetyCl}(\Phi)(w)$ defined by $\mathit{CoSafetyCl}(\Phi)(w) = \sup_{u \prec w} \inf_{w' \in \Sigma^\omega} \Phi(uw')$ for all $w \in \Sigma^\omega$.*

It is easy to see that safety and co-safety are duals in the following sense.

Theorem 5.3.10. *A property $\Phi : \Sigma^\omega \to \mathbb{D}$ is safe iff $\overline{\Phi}$ is co-safe.*

Thanks to Theorem 5.3.10, the duals of the results above for safety properties and the safety closure operation hold for co-safety properties and the co-safety closure operation. To demonstrate, let us define and investigate the *maximal response-time* property.

Example 5.3.11. *Let $\Sigma = \{\mathtt{rq}, \mathtt{gr}, \mathtt{tk}, \mathtt{oo}\}$ and $\mathbb{D} = \mathbb{N} \cup \{\infty\}$. We define the maximal response-time property $\Phi_{\max}$ through an auxiliary property that computes the current response*

time for each finite trace. In particular, for all $u \in \Sigma^*$, let $\pi_{curr}(u) = |u|_{tk} - |u'|_{tk}$, where $u' \preceq u$ is the longest prefix of u without pending rq. Then, let $\pi_{\max}(u) = \max_{u' \preceq u} \pi_{curr}(u')$ for all $u \in \Sigma^*$, and $\Phi_{\max}(w) = \lim_{u \prec w} \pi_{curr}(u)$ for all $w \in \Sigma^\omega$. The limit always exists because $\pi_{\max}$ is nondecreasing. Note the contrast between π_{curr} and π_{last} from Theorem 5.3.2. While π_{curr} takes an optimistic view of the future and assumes the gr will follow immediately, π_{last} takes a pessimistic view and assumes the gr will never follow. Now, let $w \in \Sigma^\omega$ and $v \in \mathbb{D}$. If the maximal response time of w is strictly greater than v, then for some prefix $u \prec w$ the current response time is strictly greater than v also, which means that, no matter what happens in the future, the maximal response time is strictly greater than v after observing u. Therefore, $\Phi_{\max}$ is co-safe. By a similar reasoning, the sequence of greatest lower bounds of possible prediction values over the prefixes converges to the property value. In other words, we have $\sup_{u \prec w} \inf_{w' \in \Sigma^\omega} \Phi_{\max}(uw') = \Phi_{\max}(w)$ for all $w \in \Sigma^\omega$, thus $\Phi_{\max}$ equals its co-safety closure. Now, consider the property $\overline{\Phi_{\max}}$, which maps every trace to the same value as $\Phi_{\max}$ on a value domain where the order is reversed. It is easy to see that $\overline{\Phi_{\max}}$ is safe. Finally, recall the ghost monitor from the introduction, which maintains the infimum of possible prediction values for the maximal response-time property. Since the maximal response-time property is $\inf$-closed, the output of the ghost monitor after every prefix is realizable by some future continuation, and that output is $\pi_{\max}(u) = \max_{u' \preceq u} \pi_{curr}(u')$ for all $u \in \Sigma^*$.

Although minimal and maximal response-time properties are $\sup$- and $\inf$-closed, let us note that safety and co-safety are independent of $\sup$- and $\inf$-closedness.

Proposition 5.3.12. *There is a property Φ that is safe and co-safe but neither $\sup$- nor $\inf$-closed.*

Proof. Let $\Sigma = \{a, b\}$ be an alphabet and $\mathbb{D} = \{v_1, v_2, \bot, \top\}$ be a lattice where v_1 and v_2 are incomparable. Let $\Phi(w) = v_1$ if $a \prec w$ and $\Phi(w) = v_2$ if $b \prec w$. The property Φ is safe and co-safe because after observing the first letter, we know the value of the infinite word. However, it is not $\sup$-closed since $\sup_{w \in \Sigma^\omega} \Phi(w) = \top$ but no infinite word has the value $\top$. Similarly, it is not $\inf$-closed either. □

5.3.1 Threshold Safety

In this section, we define threshold safety to connect the boolean and the quantitative settings. It turns out that quantitative safety and threshold safety coincide on totally-ordered value domains.

Definition 5.3.13 (Threshold safety)**.** *A property $\Phi : \Sigma^\omega \to \mathbb{D}$ is* threshold safe *when for every $v \in \mathbb{D}$ the boolean property $\Phi_{\geq v}$ is safe (and thus $\Phi_{\not\geq v}$ is co-safe). Equivalently, for every $w \in \Sigma^\omega$ and $v \in \mathbb{D}$ if $\Phi(w) \not\geq v$ then there exists $u \prec w$ such that for all $w' \in \Sigma^\omega$ we have $\Phi(uw') \not\geq v$.*

Definition 5.3.14 (Threshold co-safety)**.** *A property $\Phi : \Sigma^\omega \to \mathbb{D}$ is* threshold co-safe *when for every $v \in \mathbb{D}$ the boolean property $\Phi_{\not\leq v}$ is co-safe (and thus $\Phi_{\leq v}$ is safe). Equivalently, for every $w \in \Sigma^\omega$ and $v \in \mathbb{D}$ if $\Phi(w) \not\leq v$ then there exists $u \prec w$ such that for all $w' \in \Sigma^\omega$ we have $\Phi(uw') \not\leq v$.*

In general, quantitative safety implies threshold safety, but the converse need not hold with respect to partially-ordered value domains.

Proposition 5.3.15. *Every safety (resp. co-safety) property is threshold safe (resp. threshold co-safe), but not vice versa.*

Proof. Consider a property Φ over the value domain $\mathbb{D}$. Observe that for all $u \in \Sigma^*$ and all $v \in \mathbb{D}$, we have that $\sup_{w' \in \Sigma^\omega} \Phi(uw') \not\geq v$ implies $\Phi(uw) \not\geq v$ for all $w \in \Sigma^\omega$. If Φ is safe then, by definition, for every $w \in \Sigma^\omega$ and value $v \in \mathbb{D}$ if $\Phi(w) \not\geq v$, there is a prefix $u \prec w$ such that $\sup_{w' \in \Sigma^\omega} \Phi(uw') \not\geq v$. Thanks to the previous observation, for every $w \in \Sigma^\omega$ and value $v \in \mathbb{D}$ if $\Phi(w) \not\geq v$ then there exists $u \prec w$ such that $\Phi(uw') \not\geq v$ for all $w' \in \Sigma^\omega$. Hence Φ is threshold safe. Proving that co-safety implies threshold co-safety can be done similarly.

Consider the value domain $\mathbb{D} = [0,1] \cup \{x\}$ where x is such that $0 < x$ and $x < 1$, but it is incomparable with all $v \in (0,1)$, while within $[0,1]$ there is the standard order. Let Φ be a property defined over $\Sigma = \{a, b\}$ as follows: $\Phi(w) = x$ if $w = a^\omega$, $\Phi(w) = 2^{-|w|_a}$ if $w \in \Sigma^* b^\omega$, and $\Phi(w) = 0$ otherwise.

First, we show that Φ is threshold safe. Let $w \in \Sigma^\omega$ and $v \in \mathbb{D}$. If $v = x$, then $\Phi_{\geq v} = \{a^\omega, b^\omega\}$, which is safe. If $v = 0$, then $\Phi_{\geq v} = \Sigma^\omega$, which is safe as well. Otherwise, if $v \in (0,1]$, there exists $n \in \mathbb{N}$ such that the boolean property $\Phi_{\geq v}$ contains exactly the words w' such that $|w'|_a \leq n$, which is again safe. Therefore Φ is threshold safe.

Now, we show that Φ is not safe. To witness, let $w = a^\omega$ and $v \in (0,1)$. Observe that $\Phi(w) \not\geq v$. Moreover, for every prefix $u \prec w$, there exist continuations $w_1 = a^\omega$ and $w_2 = b^\omega$ such that $\Phi(uw_1) = x$ and $\Phi(uw_2) \in (0,1)$. Then, it is easy to see that for every prefix $u \prec w$ we have $\sup_{w' \in \Sigma^\omega} \Phi(uw') = 1 \geq v$. Therefore, Φ is not safe. Moreover, its complement $\overline{\Phi}$ is threshold co-safe but not co-safe. □

While safety and threshold safety can differ when considering a single fixed threshold, the two definitions are equivalent on totally-ordered domains since both inherently quantify over all thresholds.

Theorem 5.3.16. *Let $\mathbb{D}$ be a totally-ordered value domain. A property $\Phi : \Sigma^\omega \to \mathbb{D}$ is safe (resp. co-safe) iff it is threshold safe (resp. threshold co-safe).*

Proof. We prove only the safety case; the co-safety case follows by duality. Consider a property $\Phi : \Sigma^\omega \to \mathbb{D}$ where $\mathbb{D}$ is totally ordered. By Theorem 5.3.15, if Φ is safe then it is also threshold safe.

For the other direction, having that Φ is not safe, i.e., for some $w_1 \in \Sigma^\omega$ and $v_1 \in \mathbb{D}$ for which $\Phi(w_1) < v_1$, and every prefix $u_1 \prec w_1$ satisfies that $\sup_{w \in \Sigma^\omega} \Phi(u_1 w) \geq v_1$, we exhibit $w_2 \in \Sigma^\omega$ and $v_2 \in \mathbb{D}$ for which $\Phi(w_2) < v_2$, and every prefix $u_2 \prec w_2$ admits a continuation $w \in \Sigma^\omega$ such that $\Phi(u_2 w) \geq v_2$. We proceed case by case depending on how $\sup_{w \in \Sigma^\omega} \Phi(u_1 w) \geq v_1$ holds.

- Suppose $\sup_{w \in \Sigma^\omega} \Phi(u_1 w) > v_1$ for all $u_1 \prec w_1$. Then, let $w_2 = w_1$ and $v_2 = v_1$, and observe that the claim holds since the supremum is either realizable by an infinite continuation or it can be approximated arbitrarily closely.
- Suppose $\sup_{w \in \Sigma^\omega} \Phi(u_1 w) = v_1$ for some $u_1 \prec w_1$, and for every finite continuation $u_1 \preceq r \prec w_1$ there exists an infinite continuation $w' \in \Sigma^\omega$ such that $\Phi(rw') = v_1$. Then, let $w_2 = w_1$ and $v_2 = v_1$, and observe that the claim holds since the supremum is realizable by some infinite continuation.

- Suppose $\sup_{w\in\Sigma^\omega} \Phi(u_1 w) = v_1$ for some $u_1 \prec w_1$, and for some finite continuation $u_1 \preceq r \prec w_1$, every infinite continuation $w' \in \Sigma^\omega$ satisfies $\Phi(rw') < v_1$. Let $\hat{r}$ be the shortest finite continuation for which $\Phi(rw') < v_1$ for all $w' \in \Sigma^\omega$. Since $\Phi(w_1) < v_1$ and $\mathbb{D}$ is totally ordered, there exists v_2 such that $\Phi(w_1) < v_2 < v_1$. We recall that, from the nonsafety of Φ, all prefixes $u_1 \prec w_1$ satisfy $\sup_{w\in\Sigma^\omega} \Phi(u_1 w) \geq v_1 > v_2$. Then, let $w_2 = w_1$ and $\Phi(w_1) < v_2 < v_1$, and observe that the claim holds since the supremum can be approximated arbitrarily closely. □

Finally, we also show that the two definitions coincide for $\sup$-closed properties.

Proposition 5.3.17. *Let $\Phi : \Sigma^\omega \to \mathbb{D}$ be a $\sup$-closed (resp. $\inf$-closed) property. Then, Φ is safe (resp. co-safe) iff it is threshold safe (resp. threshold co-safe).*

Proof. We prove only the safety case; the co-safety case follows by duality. Consider a $\sup$-closed property $\Phi : \Sigma^\omega \to \mathbb{D}$. By Theorem 5.3.15, if Φ is safe then it is also threshold safe. For the other direction, suppose Φ is threshold safe. Let $w \in \Sigma^\omega$ and $v \in \mathbb{D}$ be such that $\Phi(w) \not\geq v$. Then, there exists $u \prec w$ such that $\Phi(uw') \not\geq v$ for all $w' \in \Sigma^\omega$. Since Φ is $\sup$-closed, there exists $\hat{w} \in \Sigma^\omega$ with $\Phi(u\hat{w}) = \sup_{w'\in\Sigma^\omega} \Phi(uw')$. Therefore, we have $\sup_{w'\in\Sigma^\omega} \Phi(uw') \not\geq v$, implying that Φ is safe. □

5.3.2 Continuity and Discounting

We move next to the relation between safety and continuity. We recall some standard definitions; more about them can be found in textbooks, e.g., [HR86, GG99, GHK+03].

A *topology* of a set X can be defined to be its collection τ of open subsets, and the pair (X, τ) stands for a *topological space*. It is *metrizable* when there exists a distance function (metric) d on X such that the topology induced by d on X is τ.

Given a topological space (X, τ), a set $S \subseteq X$ is closed in (X, τ) iff its complement $\overline{S} = X \setminus S$ is open in (X, τ). Moreover, given a set $S \subseteq X$, the topological closure $TopolCl(S)$ of S is the smallest closed set that contains S, and the topological interior $TopolInt(S)$ of S is the greatest open set that is contained in S.

The *Cantor space* of infinite words is the set Σ^ω with the metric $\mu : \Sigma^\omega \times \Sigma^\omega \to [0, 1]$ such that $\mu(w, w) = 0$ and $\mu(w, w') = 2^{-|u|}$ where $u \in \Sigma^*$ is the longest common prefix of $w, w' \in \Sigma^\omega$ with $w \neq w'$. Accordingly, a set $P \subset \Sigma^\omega$ is *open* in the Cantor space of infinite words iff for every $w \in P$ there exists a prefix $u \prec w$ such that $u\Sigma^\omega \subseteq P$.

Let $\mathbb{D}$ be a value domain and $S \subseteq \mathbb{D}$ a subset. Let $\uparrow S = \{y \in \mathbb{D} \mid \exists x \in S : x \leq y\}$ and $\downarrow S = \{y \in \mathbb{D} \mid \exists x \in S : y \leq x\}$.

A set $S \subseteq \mathbb{D}$ is *upward directed* iff for every $x, y \in S$ there is $z \in S$ such that $x \leq z$ and $y \leq z$. A set $S \subseteq \mathbb{D}$ is *Scott open* iff (i) $S = \uparrow S$, and (ii) $\sup V \in S$ implies $V \cap S \neq \emptyset$ for all upward-directed sets $V \subseteq \mathbb{D}$. A set $S \subseteq \mathbb{D}$ is *Scott closed* iff its complement $\overline{S}$ is Scott open. The *Scott topology* on a complete lattice $\mathbb{D}$ is the topology induced by the Scott open sets of $\mathbb{D}$. Considering the Scott topology on $\mathbb{D}$, we have $TopolCl(\{v\}) = \downarrow\{v\}$ for every $v \in \mathbb{D}$.

The *dual Scott topology* on $\mathbb{D}$ is the Scott topology on the inverse $\overline{\mathbb{D}}$ of $\mathbb{D}$. An equivalent definition can be obtained by using the duals of above notions as follows. A set $S \subseteq \mathbb{D}$ is *downward directed* iff for every $x, y \in S$ there is $z \in S$ such that $z \leq x$ and $z \leq y$. A set $S \subseteq \mathbb{D}$ is *dual Scott open* iff (i) $S = \downarrow S$, and (ii) $\inf V \in S$ implies $V \cap S \neq \emptyset$ for all

downward-directed sets $V \subseteq \mathbb{D}$. A set $S \subseteq \mathbb{D}$ is *dual Scott closed* iff its complement $\overline{S}$ is dual Scott open. Then, the *dual Scott topology* on a complete lattice $\mathbb{D}$ is the topology induced by the dual Scott open sets of $\mathbb{D}$. Considering the dual Scott topology on $\mathbb{D}$, we have $TopolCl(\{v\}) = \uparrow\{v\}$ for every $v \in \mathbb{D}$.

Consider a totally-ordered value domain $\mathbb{D}$. For each element $v \in \mathbb{D}$, let $L_v = \{v' \in \mathbb{D} \mid v' < v\}$ and $R_v = \{v' \in \mathbb{D} \mid v < v'\}$. The *order topology* on $\mathbb{D}$ is generated by the set $\{L_v \mid v \in \mathbb{D}\} \cup \{R_v \mid v \in \mathbb{D}\}$. Moreover, the *left order topology* (resp. *right order topology*) is generated by the set $\{L_v \mid v \in \mathbb{D}\}$ (resp. $\{R_v \mid v \in \mathbb{D}\}$).

For a given property $\Phi : \Sigma^\omega \to \mathbb{D}$ and a set $V \subseteq \mathbb{D}$ of values, the *preimage* of V on Φ is defined as $\Phi^{-1}(V) = \{w \in \Sigma^\omega \mid \Phi(w) \in V\}$. A property $\Phi : \Sigma^\omega \to \mathbb{D}$ on a topological space $\mathbb{D}$ is *continuous* when for every open subset $V \subseteq \mathbb{D}$ the preimage $\Phi^{-1}(V) \subseteq \Sigma^\omega$ is open.

In [HS21], a property Φ is defined as co-continuous when $\Phi(w) = \lim_{u \prec w} \sup_{w' \in \Sigma^\omega} \Phi(uw')$ and as continuous when $\Phi(w) = \lim_{u \prec w} \inf_{w' \in \Sigma^\omega} \Phi(uw')$ for all $w \in \Sigma^\omega$, extending the standard definitions of upper semicontinuity and lower semicontinuity for functions on extended reals to functions from infinite words to complete lattices. Co-continuity and continuity respectively coincide with safety and co-safety properties. This characterization holds because each definition is equivalent to a property expressing the same function as its corresponding closure (see Theorem 5.3.6). We complete the picture by providing a purely topological characterization of safety and co-safety properties in terms of their continuity.

Theorem 5.3.18. *Consider a property $\Phi : \Sigma^\omega \to \mathbb{D}$. If Φ is safe (resp. co-safe), then it is continuous with respect to the dual Scott topology (resp. Scott topology) on $\mathbb{D}$.*

Proof. Let $\Phi : \Sigma^\omega \to \mathbb{D}$ be a property. We prove the statement for safety properties. The case of co-safety is dual.

Assume Φ is safe. Let $S \subseteq \mathbb{D}$ be an open set and suppose towards contradiction that $\Phi^{-1}(S) \subseteq \Sigma^\omega$ is not open. There exists a word $w \in \Phi^{-1}(S)$ such that for every prefix $u \prec w$ there exists a continuation w' such that $uw' \notin \Phi^{-1}(S)$. It implies that for each such prefix u, we have $\sup_{w' \in \Sigma^\omega} \Phi(uw') \notin S$. For each $i \geq 1$, let $u_i \prec w$ be of length i, and consider the set $V = \{\sup_{w' \in \Sigma^\omega} \Phi(u_i w') \mid u_i \prec w\}$. Observe that V is a downward-directed set. If $\inf V \in S$, since S is open, we have $V \cap S \neq \emptyset$, i.e., $\sup_{w' \in \Sigma^\omega} \Phi(u_i w') \in S$ for some $u_i \prec w$. Then, we have $\Phi(u_i w') \in S$ for all $w' \in \Sigma^\omega$ since $S = {\downarrow}S$, which contradicts the supposition that $\Phi^{-1}(S) \subseteq \Sigma^\omega$ is not open. If $\inf V \notin S$, then observe that $\inf V = SafetyCl(\Phi)(w)$. Moreover, $SafetyCl(\Phi)(w) = \Phi(w)$ since Φ is safe, which implies $\inf V \in S$ since $\Phi(w) \in S$, which is a contradiction. Therefore, $\Phi^{-1}(S) \subseteq \Sigma^\omega$ is open, and thus Φ is continuous. □

The converse does not hold in general essentially due to the fact that the safety closure values may be unrealizable.

Proposition 5.3.19. *There exists a property $\Phi : \Sigma^\omega \to \mathbb{D}$ that is continuous with respect to the dual Scott topology (resp. Scott topology) on $\mathbb{D}$ but not safe (resp. co-safe).*

Proof. Let us recall the property Φ from the proof of Theorem 5.3.15: Consider the value domain $\mathbb{D} = [0, 1] \cup \{x\}$ where x is such that $0 < x$ and $x < 1$, but it is incomparable with all $v \in (0, 1)$, while within $[0, 1]$ there is the standard order. Let Φ be a property defined over $\Sigma = \{a, b\}$ as follows: $\Phi(w) = x$ if $w = a^\omega$, $\Phi(w) = 2^{-|w|_a}$ if $w \in \Sigma^* b^\omega$, and $\Phi(w) = 0$ otherwise. We showed in the proof of Theorem 5.3.15 that Φ is not safe. Below, we show

that Φ is continuous with respect to the dual Scott topology on $\mathbb{D}$. One can symmetrically show that $\overline{\Phi}$ is continuous with respect to the Scott topology on $\mathbb{D}$ but not co-safe.

Let us identify the open subsets of $\mathbb{D}$. The sets $\emptyset$ and $\mathbb{D}$ are open in $\mathbb{D}$ as they are open in any topology. Moreover, notice that every open subset containing 1 is exactly the entire value domain due to the downward closure requirement. Now, consider a subset $S \subseteq \mathbb{D}$ with $1 \notin S$. We argue that if S is open, it is either of the form $[0, r)$ or $[0, r) \cup \{x\}$ for some $r \in (0, 1]$.

First, consider the case when $x \notin S$. Notice that again due to the downward closure requirement the set S must contain an interval $I \subseteq [0, 1]$ with $0 \in I$. Moreover, the interval I cannot contain its upper bound. Suppose towards contradiction that $I = [0, r]$ for some $r \in [0, 1]$. If $r = 1$, then $S = [0, 1]$, which is not open because it violates the downward closure requirement since $x \notin S$. If $r < 1$, then $S = [0, r]$, which is not open because $V = (r, 1]$ is a downward-directed set with $\inf V = r \in S$ but $V \cap S = \emptyset$. Therefore, if $x \notin S$, then S is of the form $[0, r)$ for some $r \in (0, 1]$. For the case of $x \in S$, notice that the inclusion of x in S does not affect the downward closure requirement. Moreover, the only downward-directed sets whose infimum is x are $\{x\}$ and $\{x, 1\}$, and their intersection with S is not empty as $x \in S$. Therefore, if $x \in S$, then S is of the form $[0, r) \cup \{x\}$ for some $r \in (0, 1]$.

Now, let us show that Φ is continuous. If $S = \emptyset$ (resp. $\mathbb{D}$), then we have $\Phi^{-1}(S) = \emptyset$ (resp. Σ^ω), which is evidently open in the Cantor topology of Σ^ω. Suppose $S = [0, r)$ for some $r \in (0, 1]$. Let $k_r = \min\{k \in \mathbb{N} \mid 2^{-k} < r\}$. Then, observe that $\Phi^{-1}(S)$ is exactly the set of infinite words w where w contains at least k_r occurrences of a and at least one b, which is an intersection of two open sets, and thus open. Finally, suppose $S = [0, r) \cup \{x\}$ for some $r \in (0, 1]$. Let k_r be as above, and notice that $\Phi^{-1}(S)$ is exactly the set of infinite words w where w contains at least k_r occurrences of a, which is open. Therefore, Φ is continuous. □

Next, we examine the relation between threshold safety and continuity with respect to the dual Scott topology. We show in particular that continuity implies threshold safety.

Theorem 5.3.20. *Consider a property $\Phi : \Sigma^\omega \to \mathbb{D}$. If Φ is continuous with respect to the dual Scott topology (resp. Scott topology) on $\mathbb{D}$, then it is threshold safe (resp. threshold co-safe).*

Proof. Let $\Phi : \Sigma^\omega \to \mathbb{D}$ be a property. We prove the statement for safety properties. The case of co-safety is dual.

Assume Φ is continuous, i.e., for every open set $S \subseteq \mathbb{D}$ the preimage $\Phi^{-1}(S)$ is open. We want to show that Φ is threshold safe, i.e., $\Phi_{\not\geq v} = \{w \in \Sigma^\omega \mid \Phi(w) \not\geq v\}$ is co-safe in the boolean sense for every $v \in \mathbb{D}$. Let $v \in \mathbb{D}$ and notice that $\Phi_{\not\geq v} = \Phi^{-1}(\overline{\uparrow\{v\}})$. Since the set $\overline{\uparrow\{v\}}$ is open in $\mathbb{D}$ and Φ is continuous, its preimage $\Phi_{\not\geq v}$ is open in Σ^ω, i.e., co-safe in the boolean sense. Therefore, Φ is threshold safe. □

Moreover, we establish that the inclusion is strict: there is a threshold safety property that is not continuous with respect to the dual Scott topology.

Proposition 5.3.21. *There exists a property $\Phi : \Sigma^\omega \to \mathbb{D}$ that is threshold safe (resp. threshold co-safe) but not continuous with respect to the dual Scott topology (resp. Scott topology) on $\mathbb{D}$.*

Proof. Let $\Sigma = \{a, b\}$ be a finite alphabet. Consider the value domain $\mathbb{D} = \Sigma^\omega \cup \{\bot, \top\}$ where for every $x \in \mathbb{D}$ we have $\top \geq x$ and $x \geq \bot$, but the elements from Σ^ω are incomparable with each other. Let $\Phi : \Sigma^\omega \to \mathbb{D}$ be such that $\Phi(w) = w$ for all $w \in \Sigma^\omega$.

We show that Φ is threshold safe, i.e., for every $w \in \Sigma^\omega$ and every $v \in \mathbb{D}$ with $\Phi(w) \not\geq v$ there exists $u \prec w$ such that for every $w' \in \Sigma^\omega$ we have $\Phi(uw') \not\geq v$. Let $w \in \Sigma^\omega$ and $v \in \mathbb{D}$. If $v = \top$, the finite witness for $\Phi(w) \not\geq \top$ is the empty word since no infinite word has the value $\top$. If $v < \top$, we have $\Phi(w) \not\geq v$ iff $v \in \Sigma^\omega$ and $w \neq v$ since Φ is the identity function on Σ^ω and the elements from Σ^ω are incomparable. Observe that two infinite words are distinct iff there is a finite word that is a prefix of one and not the other. Then, such a prefix of w is the finite witness for $\Phi(w) \not\geq v$. Therefore, Φ is threshold safe.

We show that Φ is not continuous with respect to the dual Scott topology on $\mathbb{D}$. Let $P \subseteq \Sigma^\omega$ be a set of infinite words. First, we argue that $S = P \cup \{\bot\}$ is open in $\mathbb{D}$. The set S is downward closed because for every $w \in P$ the only element smaller than w is $\bot$, which is in S. Let V be a downward-directed subset of $\mathbb{D}$. If $\bot \in V$, then $\inf V = \bot \in S$ and we have $V \cap S \neq \emptyset$. If $\bot \notin V$, then V contains at most one element from Σ^ω (otherwise we would have $\bot \in V$ since V is downward directed). If V contains no elements from Σ^ω, then it is either $\emptyset$ or $\{\top\}$, and thus $\inf V = \top \notin S$. If V contains some element w from Σ^ω, we have $\inf V = w$. Moreover, if $\inf V \in S$, then clearly $w \in S$ and thus $V \cap S \neq \emptyset$.

Now, let $P = \Sigma^* a^\omega$. As we proved above, the set $S = P \cup \{\bot\}$ is open in $\mathbb{D}$. However, its preimage $\Phi^{-1}(S)$ is exactly the set P, which is not open in Σ^ω. Therefore, Φ is not continuous.

The property Φ above and the same arguments also cover the case of co-safety. □

An immediate result of Theorems 5.3.18 and 5.3.20 is that whenever safety and threshold safety coincide, they also coincide with continuity with respect to the dual Scott topology. In particular, thanks to Theorem 5.3.17, we obtain the following.

Corollary 5.3.22. *Consider a* $\sup$*-closed (resp.* $\inf$*-closed) property* $\Phi : \Sigma^\omega \to \mathbb{D}$*. Then,* Φ *is safe (resp. co-safe) iff it is continuous with respect to the dual Scott topology (resp. Scott topology) on* $\mathbb{D}$*.*

Moreover, for totally-ordered value domains $\mathbb{D}$, it is well known that a property is continuous with respect to the dual Scott topology (resp. Scott topology) on $\mathbb{D}$ iff it is continuous with respect to the left order topology (resp. right order topology) on $\mathbb{D}$, which coincides with upper semicontinuity (resp. lower semicontinuity) when $\mathbb{D} = \mathbb{R} \cup \{-\infty, +\infty\}$. Then, thanks to Theorem 5.3.16, we get the following.

Corollary 5.3.23. *Let* $\mathbb{D}$ *be a totally-ordered value domain. A property* $\Phi : \Sigma^\omega \to \mathbb{D}$ *is safe (resp. co-safe) iff it is continuous with respect to the left order topology (resp. right order topology) on* $\mathbb{D}$*.*

Finally, since a property is continuous with respect to the order topology on $\mathbb{D}$ iff it is continuous with respect to both left and right order topologies on $\mathbb{D}$, we immediately obtain the following.

Corollary 5.3.24. *Let* $\mathbb{D}$ *be a totally-ordered value domain. A property* $\Phi : \Sigma^\omega \to \mathbb{D}$ *is safe and co-safe iff it is continuous with respect to the order topology on* $\mathbb{D}$*.*

Now, we shift our focus to totally-ordered value domains whose order topology is metrizable. We provide a general definition of discounting properties on such domains.

Definition 5.3.25 (Discounting). *Let $\mathbb{D}$ be a totally-ordered value domain for which the order topology is metrizable with a metric d. A property $\Phi : \Sigma^\omega \to \mathbb{D}$ is* discounting *when for every $\varepsilon > 0$ there exists $n \in \mathbb{N}$ such that for every $u \in \Sigma^n$ and $w, w' \in \Sigma^\omega$ we have $d(\Phi(uw), \Phi(uw')) < \varepsilon$.*

Intuitively, a property is discounting when the range of potential values for every word converges to a singleton. As an example, consider the following discounted safety property: Given a boolean safety property P, let Φ be a quantitative property such that $\Phi(w) = 1$ if $w \in P$, and $\Phi(w) = 2^{-|u|}$ if $w \notin P$, where $u \prec w$ is the shortest bad prefix of w for P. We remark that our definition captures the previous definitions of discounting given in [dAHM03, ABK14].

Remark 5.3.26. *Notice that the definition of discounting coincides with uniform continuity. Since Σ^ω equipped with Cantor distance is a compact space, every continuous property is also uniformly continuous by Heine-Cantor theorem, and thus discounting.*

As an immediate consequence, we obtain the following.

Corollary 5.3.27. *Let $\mathbb{D}$ be a totally-ordered value domain for which the order topology is metrizable. A property $\Phi : \Sigma^\omega \to \mathbb{D}$ is safe and co-safe iff it is discounting.*

Let $P \subseteq \Sigma^\omega$ be a boolean property. Recall that $TopolCl(P)$ is the smallest boolean safety property that contains P, and $TopolInt(P)$ of P is the greatest boolean co-safety property that is contained in P. To conclude this subsection, we show the connection between the quantitative safety closure (resp. co-safety closure) and the topological closure (resp. topological interior) through $\sup$-closedness (resp. $\inf$-closedness). The $\sup$-closedness assumption makes the quantitative safety closure values realizable. This guarantees that for every value v, every word whose safety closure value is at least v belongs to the topological closure of the set of words whose property values are at least v. Similarly, the $\inf$-closedness assumption helps in the case of co-safety and topological interior.

Theorem 5.3.28. *Consider a property $\Phi : \Sigma^\omega \to \mathbb{D}$ and a threshold $v \in \mathbb{D}$. If Φ is* $\sup$*-closed, then $(SafetyCl(\Phi))_{\geq v} = TopolCl(\Phi_{\geq v})$. If Φ is* $\inf$*-closed, then $(CoSafetyCl(\Phi))_{\leq v} = TopolInt(\Phi_{\leq v})$.*

Proof. First, we observe that for all $u \in \Sigma^*$, if $\sup_{w' \in \Sigma^\omega} \Phi(uw') \not\geq v$ then for every $w \in \Sigma^\omega$, we have $\Phi(uw) \not\geq v$. Next, we show that $TopolCl(\Phi_{\geq v}) \subseteq (SafetyCl(\Phi))_{\geq v}$. Suppose towards contradiction that there exists $w \in TopolCl(\Phi_{\geq v}) \setminus (SafetyCl(\Phi))_{\geq v}$, that is, $SafetyCl(\Phi)(w) \not\geq v$ and $w \in TopolCl(\Phi_{\geq v})$. This means that (i) $\inf_{u \prec w} \sup_{w' \in \Sigma^\omega} \Phi(uw') \not\geq v$, and (ii) for every prefix $u \prec w$ there exists $w' \in \Sigma^\omega$ such that $\Phi(uw') \geq v$. By the above observation, (i) implies that there exists a prefix $u' \prec w$ such that for all $w'' \in \Sigma^\omega$ we have $\Phi(u'w'') \not\geq v$, which contradicts (ii).

Now, we show that if Φ is $\sup$-closed then $(SafetyCl(\Phi))_{\geq v} \subseteq TopolCl(\Phi_{\geq v})$. Suppose towards contradiction that there exists $w \in (SafetyCl(\Phi))_{\geq v} \setminus TopolCl(\Phi_{\geq v})$, that is, $SafetyCl(\Phi)(w) \geq v$ and $w \notin TopolCl(\Phi_{\geq v})$. By the duality between closure and interior, we have $w \in TopolInt(\Phi_{\not\geq v})$. Then, (i) $\inf_{u \prec w} \sup_{w' \in \Sigma^\omega} \Phi(uw') \geq v$, and (ii) there exists $u' \prec w$ such that for all $w'' \in \Sigma^\omega$ we have $\Phi(u'w'') \not\geq v$. Since Φ is $\sup$-closed, (i) implies that for every prefix $u \prec w$ there exists $w' \in \Sigma^\omega$ such that $\Phi(uw') \geq v$, which contradicts (ii).

Proving that if Φ is $\inf$-closed then $(CoSafetyCl(\Phi))_{\leq v} = TopolInt(\Phi_{\leq v})$ can be done similarly, based on the observation that for all $u \in \Sigma^*$, if $\inf_{w' \in \Sigma^\omega} \Phi(uw') \not\leq v$ then for every word $w \in \Sigma^\omega$, we have $\Phi(uw) \not\leq v$. □

5.3.3 Additional Notions Related to Quantitative Safety

In [LDL17], the authors consider the model-checking problem for properties on multi-valued truth domains. They introduce the notion of multi-safety through a closure operation that coincides with our safety closure. Formally, a property Φ is *multi-safe* iff $\Phi(w) = \mathit{SafetyCl}(\Phi)(w)$ for every $w \in \Sigma^\omega$. By Theorem 5.3.6, we immediately obtain the following.

Proposition 5.3.29. *A property is multi-safe iff it is safe.*

Although the two definitions of safety are equivalent, our definition is consistent with the membership problem for quantitative properties and motivated by their monitoring.

In [GS22], the authors extend a refinement of the safety-liveness classification for monitoring [PH18] to richer domains. They introduce the notion of verdict-safety through dismissibility of values not less than or equal to the property value. Formally, a property Φ is *verdict-safe* iff for every $w \in \Sigma^\omega$ and $v \not\leq \Phi(w)$, there exists a prefix $u \prec w$ such that for all $w' \in \Sigma^\omega$, we have $\Phi(uw') \neq v$.

We demonstrate that verdict-safety is weaker than safety. Moreover, we provide a condition under which the two definitions coincide. To achieve this, we reason about sets of possible prediction values: for a property Φ and $u \in \Sigma^*$, let $P_{\Phi,u} = \{\Phi(uw) \mid w \in \Sigma^\omega\}$.

Lemma 5.3.30. *A property Φ is verdict-safe iff $\Phi(w) = \sup(\lim_{u \prec w} P_{\Phi,u})$ for all $w \in \Sigma^\omega$.*

Proof. For all $w \in \Sigma^\omega$ let us define $P_w = \lim_{u \prec w} P_{\Phi,u} = \bigcap_{u \prec w} P_{\Phi,u}$. Assume Φ is verdict-safe and suppose towards contradiction that $\Phi(w) \neq \sup P_w$ for some $w \in \Sigma^\omega$. If $\Phi(w) \not\leq \sup P_w$, then $\Phi(w) \notin P_w$, which is a contradiction. Otherwise, if $\Phi(w) < \sup P_w$, there exists $v \not\leq \Phi(w)$ with $w \in P_w$. It means that there is no $u \prec w$ that dismisses the value $v \not\leq \Phi(w)$, which contradicts the fact that Φ is verdict-safe. Therefore, $\Phi(w) = \sup P_w$ for all $w \in \Sigma^\omega$.

We prove the other direction by contrapositive. Assume Φ is not verdict-safe, i.e., for some $w \in \Sigma^\omega$ and $v \not\leq \Phi(w)$, every $u \prec w$ has an extension $w' \in \Sigma^\omega$ with $\Phi(uw') = v$. Equivalently, for some $w \in \Sigma^\omega$ and $v \not\leq \Phi(w)$, every $u \prec w$ satisfies $v \in P_{\Phi,u}$. Then, $v \in P_w$, but since $v \not\leq \Phi(w)$, we have $\sup P_w > \Phi(w)$. □

Notice that Φ is safe iff $\Phi(w) = \lim_{u \prec w}(\sup P_{\Phi,u})$ for all $w \in \Sigma^\omega$, thanks to Theorem 5.3.6. Below we describe a property that is verdict-safe but not safe.

Example 5.3.31. *Let $\Sigma = \{a, b\}$. Define Φ by $\Phi(w) = 0$ if $w = a^\omega$, and $\Phi(w) = |u|$ otherwise, where $u \prec w$ is the shortest prefix in which b occurs. The property Φ is verdict-safe. First, observe that $\mathbb{D} = \mathbb{N} \cup \{\infty\}$. Let $w \in \Sigma^\omega$ and $v \in \mathbb{D}$ with $v > \Phi(w)$. If $\Phi(w) > 0$, then w contains b, and $\Phi(w) = |u|$ for some $u \prec w$ in which b occurs for the first time. After the prefix u, all $w' \in \Sigma^\omega$ yield $\Phi(uw') = |u|$, thus all values above $|u|$ are rejected. If $\Phi(w) = 0$, then $w = a^\omega$. Let $v \in \mathbb{N}$ with $v > 0$, and consider the prefix $a^v \prec w$. Observe that the set of possible prediction values after reading a^v is $\{0, v+1, v+2, \ldots\}$, therefore a^v allows the ghost monitor to reject the value v. However, Φ is not safe because, although $\Phi(a^\omega) = 0$, for every $u \prec a^\omega$, we have $\sup_{w' \in \Sigma^\omega} \Phi(uw') = \infty$.*

The separation is due to the fact that for some finite traces, the $\sup$ of possible prediction values cannot be realized by any future. This is not the case for the minimal response-time property $\Phi_{\min}$ from Theorem 5.3.2 because for every $u \in \Sigma^*$ the continuation gr^ω realizes the value $\sup_{w' \in \Sigma^\omega} \Phi_{\min}(uw')$, and thus $\Phi_{\min}$ is $\sup$-closed.

Recall from the introduction the ghost monitor that maintains the $\sup$ of possible prediction values. For monitoring $\sup$-closed properties this suffices; otherwise the ghost monitor also needs to maintain whether or not the supremum of the possible prediction values is realizable by some future continuation. In general, we have the following for every $\sup$-closed property.

Lemma 5.3.32. *Let Φ be a $\sup$-closed property. Then, $\lim_{u \prec w}(\sup P_{\Phi,u}) = \sup(\lim_{u \prec w} P_{\Phi,u})$ for all $w \in \Sigma^\omega$.*

Proof. Note that $\lim_{u \prec w}(\sup P_{\Phi,u}) \geq \sup(\lim_{u \prec w} P_{\Phi,u})$ holds in general, and we want to show that $\lim_{u \prec w}(\sup P_{\Phi,u}) \leq \sup(\lim_{u \prec w} P_{\Phi,u})$ holds for every $\sup$-closed Φ. Let $w \in \Sigma^\omega$. Since the sequence $(P_{\Phi,u})_{u \prec w}$ of sets is nonincreasing and $\sup P_{\Phi,u} \in P_{\Phi,u}$ for every $u \in \Sigma^*$ (thanks to $\sup$-closedness of Φ), we have $\sup P_{\Phi,u'} \in P_{\Phi,u}$ for every $u, u' \in \Sigma^*$ with $u \preceq u'$. Moreover, $\lim_{u \prec w}(\sup P_{\Phi,u}) \in P_{\Phi,u'}$ for every $u' \in \Sigma^*$ with $u' \prec w$. Then, by definition, we have $\lim_{u \prec w}(\sup P_{\Phi,u}) \in \lim_{u \prec w} P_{\Phi,u}$, and therefore $\lim_{u \prec w}(\sup P_{\Phi,u}) \leq \sup(\lim_{u \prec w} P_{\Phi,u})$. □

As a consequence of the above, we get the following.

Theorem 5.3.33. *Every safety property is verdict-safe, but not vice versa. Moreover, a $\sup$-closed property is safe iff it is verdict-safe.*

Let us conclude with a remark on the form of hypotheses in our definition of safety.

Remark 5.3.34. *Suppose we define safety with strict lower bound hypotheses instead of nonstrict: for every $w \in \Sigma^\omega$ and value $v \in \mathbb{D}$ with $\Phi(w) \not> v$, there is a prefix $u \prec w$ such that $\sup_{w' \in \Sigma^\omega} \Phi(uw') \not> v$. Let w be an arbitrary word and consider $v = \Phi(w)$. It is clear that this definition would require the $\sup$ of possible prediction values to converge to $\Phi(w)$ after a finite prefix, which is too restrictive.*

5.4 The Quantitative Safety-Progress Hierarchy

The safety-progress classification of boolean properties [CMP93] is a Borel hierarchy built from the Cantor topology of traces. Safety and co-safety properties lie on the first level, respectively corresponding to closed sets and open sets. The second level is obtained through countable unions and intersections of properties from the first level: persistence properties are countable unions of closed sets, while response properties are countable intersections of open sets. We generalize this construction to the quantitative setting.

In the boolean case, each property class is defined through an operation that takes a set $S \subseteq \Sigma^*$ of finite traces and produces a set $P \subseteq \Sigma^\omega$ of infinite traces. For example, to obtain a co-safety property from $S \subseteq \Sigma^*$, the corresponding operation yields $S\Sigma^\omega$. Similarly, we formalize each property class by a value function.

Definition 5.4.1 (Limit property)**.** *A property $\Phi : \Sigma^\omega \to \mathbb{D}$ is a* limit property *when there exists a finitary property $\pi : \Sigma^* \to \mathbb{D}$ and a value function* Val $: \mathbb{D}^\omega \to \mathbb{D}$ *such that $\Phi(w) = \mathsf{Val}_{u \prec w} \pi(u)$ for all $w \in \Sigma^\omega$. We denote this by $\Phi = (\pi, \mathsf{Val})$. In particular, if $\Phi = (\pi, \mathsf{Val})$ for* Val $\in \{\mathsf{Inf}, \mathsf{Sup}, \mathsf{LimInf}, \mathsf{LimSup}\}$*, then Φ is a* Val*-property.*

Remark 5.4.2. *Every quantitative property $\Phi : \Sigma^\omega \to \mathbb{D}$ where $|\Sigma| \leq |\mathbb{D}|$ is a limit property because π can encode infinite words through their prefixes and* Val *can map each infinite sequence (corresponding to a unique infinite word) to the desired value. Below, we focus on particular value functions (namely* Inf, Sup, LimInf, LimSup*) for which this is not possible.*

To account for the value functions that construct the first two levels of the safety-progress hierarchy, we start our investigation with Inf- and Sup-properties and later focus on LimInf- and LimSup- properties.

5.4.1 Infimum and Supremum Properties

Let us start by showing that Inf-properties are closed under countable infima.

Proposition 5.4.3. *Every countable infimum of* Inf*-properties is an* Inf*-property.*

Proof. Let $\Phi_i = (\pi_i, \mathsf{Inf})$ be for each $i \in \mathbb{N}$. Let $\Phi = (\pi, \mathsf{Inf})$ where $\pi(u) = \inf_{i\in\mathbb{N}} \pi_i(u)$ for all $u \in \Sigma^*$. Let $w \in \Sigma^\omega$ be arbitrary. We have $\Phi(w) = \mathsf{Inf}_{u\prec w} \inf_{i\in\mathbb{N}} \pi_i(u) = \inf_{i\in\mathbb{N}} \mathsf{Inf}_{u\prec w}\pi_i(u) = \inf_{i\in\mathbb{N}} \Phi_i(w)$.

We show below that $\mathsf{Inf}_{u\prec w} \inf_{i\in\mathbb{N}} \pi_i(u) = \inf_{i\in\mathbb{N}} \mathsf{Inf}_{u\prec w}\pi_i(u)$ holds. Note that we can assume without loss of generality that for each $i \in \mathbb{N}$, the finitary property π_i is nonincreasing. For each $i \in \mathbb{N}$, let $x_i = \mathsf{Inf}_{u\prec w}\pi_i(u)$. For each $u \prec w$, let $y_{|u|} = \inf_{i\in\mathbb{N}} \pi_i(u)$. Moreover, let $x = \inf_{i\in\mathbb{N}} x_i$ and $y = \inf_{j\in\mathbb{N}} y_j$. Let us denote by u_j the prefix of w of length j. For all $i, j \in \mathbb{N}$, we have $x \leq x_i \leq \pi_i(u_j)$ and $y \leq y_j \leq \pi_i(u_j)$. Then, x and y are lower bounds on the set $P = \{\pi_i(u_j) \mid i, j \in \mathbb{N}\}$. Now, let z be another lower bound, i.e., $z \leq \pi_i(u_j)$ for all $i, j \in \mathbb{N}$. For a fixed $i \in \mathbb{N}$, we still have $z \leq \pi_i(u_j)$ for all $j \in \mathbb{N}$. It means that z is a lower bound on the sequence $(\pi_i(u))_{u\prec w}$ and since x_i is the infimum of this sequence, we have $z \leq x_i$. Moreover, since this holds for any $i \in \mathbb{N}$ and $x = \inf_{i\in\mathbb{N}} x_i$, we have $z \leq x$. By similar arguments, we obtain $z \leq y$. It implies that both x and y are the greatest lower bound on P, which means $x = y$ due to the uniqueness of greatest lower bound. □

Next, we demonstrate that the minimal response-time property is an Inf-property.

Example 5.4.4. *Recall the safety property $\Phi_{\min}$ of minimal response time from Theorem 5.3.2. We can equivalently define $\Phi_{\min}$ as a limit property by taking the finitary property π_{last} and the value function* Inf*. As discussed in Theorem 5.3.2, the function π_{last} outputs the response time for the last request when all requests are granted, and ∞ when there is a pending request or no request. Then $\mathsf{Inf}_{u\prec w}\pi_{\mathit{last}}(u) = \Phi_{\min}(w)$ for all $w \in \Sigma^\omega$, and therefore $\Phi_{\min} = (\pi_{\mathit{last}}, \mathsf{Inf})$.*

In fact, the safety properties coincide with Inf-properties.

Theorem 5.4.5. *A property Φ is safe iff it is an* Inf*-property.*

Proof. Assume Φ is safe. By Theorem 5.3.6, we have $\Phi(w) = \inf_{u\prec w} \sup_{w'\in\Sigma^\omega} \Phi(uw')$ for all $w \in \Sigma^\omega$. Then, simply taking $\pi(u) = \sup_{w'\in\Sigma^\omega} \Phi(uw')$ for all $u \in \Sigma^*$ yields that Φ is an Inf-property.

Now, assume Φ is an Inf-property, and suppose towards contradiction that Φ is not safe. In other words, let $\Phi = (\pi, \mathsf{Inf})$ for some finitary property $\pi : \Sigma^* \to \mathbb{D}$ and suppose $\mathsf{Inf}_{u\prec x} \sup_{w'\in\Sigma^\omega} \Phi(uw') > \Phi(x) = \mathsf{Inf}_{u\prec x}\pi(u)$ for some $x \in \Sigma^\omega$. Let $u \in \Sigma^*$ and note that $\sup_{w'\in\Sigma^\omega} \Phi(uw') = \sup_{w'\in\Sigma^\omega}(\mathsf{Inf}_{u'\prec uw'}\pi(u'))$ by definition. Moreover, for every $w' \in \Sigma^\omega$, notice that $\mathsf{Inf}_{u'\prec uw'}\pi(u') \leq \pi(u)$ since $u \prec uw'$. Then, we obtain $\sup_{w'\in\Sigma^\omega} \Phi(uw') \leq \pi(u)$ for every $u \in \Sigma^*$. In particular, this is also true for all $u \prec x$. Therefore, we get $\mathsf{Inf}_{u\prec x} \sup_{w'\in\Sigma^\omega} \Phi(uw') \leq \mathsf{Inf}_{u\prec x}\pi(u)$, which contradicts to our initial supposition. □

Notice that Theorems 5.4.3 and 5.4.5 imply a stronger closure result than Theorem 5.3.4: safety properties are closed under countable infima.

Defining the minimal response-time property as a limit property, we observe the following relation between its behavior on finite traces and infinite traces.

Example 5.4.6. *Consider the property $\Phi_{\min} = (\pi_{last}, \mathsf{Inf})$ from Theorem 5.4.4. Let $w \in \Sigma^\omega$ and $v \in \mathbb{D}$. Observe that if the minimal response time of w is at least v, then the last response time for each prefix $u \prec w$ is also at least v. Conversely, if the minimal response time of w is below v, then there is a prefix $u \prec w$ for which the last response time is also below v.*

In light of this observation, we provide another characterization of safety properties, explicitly relating the specified behavior of the limit property on finite and infinite traces.

Theorem 5.4.7. *A property $\Phi : \Sigma^\omega \to \mathbb{D}$ is safe iff $\Phi = (\pi, \mathsf{Val})$ such that for every $w \in \Sigma^\omega$ and value $v \in \mathbb{D}$, we have $\Phi(w) \geq v$ iff $\pi(u) \geq v$ for all $u \prec w$.*

Proof. Assume Φ is safe. Then, we know by Theorem 5.4.5 that Φ is an Inf-property, i.e., $\Phi = (\pi, \mathsf{Inf})$ for some finitary property $\pi : \Sigma^* \to \mathbb{D}$, and thus a limit property. Suppose towards contradiction that for some $w \in \Sigma^\omega$ and $v \in \mathbb{D}$ we have (i) $\Phi(w) \geq v$ and $\pi(u) \ngeq v$ for some $u \prec w$, or (ii) $\Phi(w) \ngeq v$ and $\pi(u) \geq v$ for every $u \prec w$. One can easily verify that (i) yields a contradiction, since if for some $u \prec w$ we have $\pi(u) \ngeq v$ then $\mathsf{Inf}_{u \prec w} \pi(u) = \Phi(w) \ngeq v$. Similarly, (ii) also yields a contradiction, since if $\Phi(w) = \mathsf{Inf}_{u \prec w} \pi(u) \ngeq v$ then there exists $u \prec w$ such that $\pi(u) \ngeq v$.

Now, assume $\Phi = (\pi, \mathsf{Val})$ for some finitary property π and value function Val such that for every $w \in \Sigma^\omega$ and value $v \in \mathbb{D}$ we have $\Phi(w) \geq v$ iff $\pi(u) \geq v$ for every $u \prec w$. We claim that $\Phi(w) = \mathsf{Inf}_{u \prec w} \pi(u)$ for every $w \in \Sigma^\omega$. Suppose towards contradiction that the equality does not hold for some trace. If $\Phi(w) \ngeq \mathsf{Inf}_{u \prec w} \pi(u)$ for some $w \in \Sigma^\omega$, let $v = \mathsf{Inf}_{u \prec w} \pi(u)$ and observe that (i) $\Phi(w) \ngeq v$, and (ii) $\mathsf{Inf}_{u \prec w} \pi(u) \geq v$. However, while (i) implies $\pi(u) \ngeq v$ for some $u \prec w$ by hypothesis, (ii) implies $\pi(u) \geq v$ for all $u \prec w$, resulting in a contradiction. The case where $\Phi(w) \nleq \mathsf{Inf}_{u \prec w} \pi(u)$ for some $w \in \Sigma^\omega$ is similar. It means that Φ is an Inf-property. Therefore, Φ is safe by Theorem 5.4.5. □

Finally, observe that the maximal response-time property is a Sup-property. As Sup-properties and Inf-properties are dual, Sup-properties are closed under countable suprema (see Theorem 5.4.3). Thanks to the duality between safety and co-safety, we also obtain the following characterizations.

Theorem 5.4.8. *For every property $\Phi : \Sigma^\omega \to \mathbb{D}$, the following are equivalent.*

1. *Φ is co-safe.*
2. *Φ is a* Sup*-property.*
3. *$\Phi = (\pi, \mathsf{Val})$ such that for every $w \in \Sigma^\omega$ and value $v \in \mathbb{D}$, we have $\Phi(w) \leq v$ iff $\pi(u) \leq v$ for all $u \prec w$.*

5.4.2 Limit Inferior and Limit Superior Properties

Let us start with an observation on the minimal response-time property.

Example 5.4.9. *Recall once again the minimal response-time property $\Phi_{\min}$ from Theorem 5.3.2. In the previous subsection, we presented an alternative definition of $\Phi_{\min}$ to establish that it is an* Inf*-property. Observe that there is yet another equivalent definition of $\Phi_{\min}$ which takes the nonincreasing finitary property $\pi_{\min}$ from Theorem 5.3.2 and pairs it with either the value function* LimInf*, or with* LimSup*. Hence $\Phi_{\min}$ is both a* LimInf- *and a* LimSup-*property.*

Before moving on to investigating LimInf- and LimSup-properties more closely, we show that the above observation can be generalized.

Theorem 5.4.10. *For each* Val $\in \{\mathsf{Inf}, \mathsf{Sup}\}$*, every* Val-*property is both a* LimInf- *and a* LimSup-*property.*

Proof. Let $\Phi = (\pi, \mathsf{Inf})$ and define an alternative finitary property as follows: $\pi'(u) = \min_{u' \preceq u} \pi(u)$. One can confirm that π' is nonincreasing and thus $\lim_{u \prec w} \pi'(u) = \mathsf{Inf}_{u \prec w} \pi(u)$ for every $w \in \Sigma^\omega$. Then, letting $\Phi_1 = (\pi', \mathsf{LimInf})$ and $\Phi_2 = (\pi', \mathsf{LimSup})$, we obtain that $\Phi(w) = \Phi_1(w) = \Phi_2(w)$ for all $w \in \Sigma^\omega$. For Val = Sup we use $\max$ instead of $\min$. □

An interesting response-time property beyond safety and co-safety arises when we remove extreme values: instead of minimal response time, consider the property that maps every trace to a value that bounds from below, not all response times, but all of them from a point onward (i.e., all but finitely many). We call this property *tail-minimal response time*.

Example 5.4.11. *Let $\Sigma = \{rq, gr, tk, oo\}$ and π_{last} be the finitary property from Theorem 5.3.2 that computes the last response time. We define the tail-minimal response-time property as $\Phi_{tmin} = (\pi_{last}, \mathsf{LimInf})$. Intuitively, it maps each trace to the least response time over all but finitely many requests. This property is interesting as a performance measure, because it focuses on the long-term performance by ignoring finitely many outliers. Consider $w \in \Sigma^\omega$ and $v \in \mathbb{D}$. Observe that if the tail-minimal response time of w is at least v, then there is a prefix $u \prec w$ such that for all longer prefixes $u \preceq u' \prec w$, the last response time in u' is at least v, and vice versa.*

Similarly as for Inf-properties, we characterize LimInf-properties through a relation between property behaviors on finite and infinite traces.

Theorem 5.4.12. *A property $\Phi : \Sigma^\omega \to \mathbb{D}$ is a* LimInf-*property iff $\Phi = (\pi, \mathsf{Val})$ such that for every $w \in \Sigma^\omega$ and value $v \in \mathbb{D}$, we have $\Phi(w) \geq v$ iff there exists $u \prec w$ such that for all $u \preceq u' \prec w$, we have $\pi(u') \geq v$.*

Proof. Assume Φ is a LimInf-property, i.e., $\Phi = (\pi, \mathsf{LimInf})$ for some finitary property $\pi : \Sigma^* \to \mathbb{D}$. Suppose towards contradiction that for some $w \in \Sigma^\omega$ and $v \in \mathbb{D}$ we have (i) $\Phi(w) \geq v$ and for all $u \prec w$ there exists $u \preceq u' \prec w$ such that $\pi(u') \not\geq v$, or (ii) $\Phi(w) \not\geq v$ and there exists $u \prec w$ such that for all $u \preceq u' \prec w$ we have $\pi(u') \geq v$. One can easily verify that (i) yields a contradiction, since if for all $u \prec w$ there exists $u \preceq u' \prec w$ with $\pi(u') \not\geq v$, then $\mathsf{LimInf}_{u \prec w} \pi(u) = \Phi(w) \not\geq v$. Similarly, (ii) also yields a contradiction, since if there exists $u \prec w$ such that for all $u \preceq u' \prec w$ we have $\pi(u') \geq v$ then $\mathsf{LimInf}_{u \prec w} \pi(u) = \Phi(w) \geq v$.

Now, assume $\Phi = (\pi, \mathsf{Val})$ for some finitary property π and value function Val such that for every $w \in \Sigma^\omega$ and value $v \in \mathbb{D}$ we have $\Phi(w) \geq v$ iff there exists $u \prec w$ such that for all $u \preceq u' \prec w$ we have $\pi(u') \geq v$. We claim that $\Phi(w) = \mathsf{LimInf}_{u \prec w} \pi(u)$ for every $w \in \Sigma^\omega$. Suppose towards contradiction that the equality does not hold for some trace. If $\Phi(w) \not\geq \mathsf{LimInf}_{u \prec w} \pi(u)$ for some $w \in \Sigma^\omega$, let $v = \mathsf{LimInf}_{u \prec w} \pi(u)$ and observe that (i) $\Phi(w) \not\geq v$, and (ii) $\mathsf{LimInf}_{u \prec w} \pi(u) \geq v$. However, by hypothesis, (i) implies that for all $u \prec w$ there exists $u \preceq u' \prec w$ with $\pi(u') \not\geq v$, which means that $\mathsf{LimInf}_{u \prec w} \pi(u) \not\geq v$, resulting in a contradiction to (ii). The case where $\Phi(w) \not\leq \mathsf{LimInf}_{u \prec w} \pi(u)$ for some $w \in \Sigma^\omega$ is similar. Therefore, Φ is a LimInf-property. □

Next, we show that LimInf-properties are closed under pairwise minimum.

Proposition 5.4.13. *For every value domain $\mathbb{D}$, the set of* LimInf*-properties over $\mathbb{D}$ is closed under* $\min$.

Proof. Consider two LimInf-properties $\Phi_1 = (\pi_1, \mathsf{LimInf})$, $\Phi_2 = (\pi_2, \mathsf{LimInf})$ and let Φ be as follows: $\Phi = (\pi, \mathsf{LimInf})$ where $\pi(u) = \min(\pi_1(u), \pi_2(u))$ for all $u \in \Sigma^*$. We now prove that $\Phi(w) = \min(\Phi_1(w), \Phi_2(w))$ for all $w \in \Sigma^\omega$.

Suppose towards contradiction that $\min(\Phi_1(w), \Phi_2(w)) \not\geq \Phi(w)$ for some $w \in \Sigma^\omega$. Observe that for all $w' \in \Sigma^\omega$ and $v \in \mathbb{D}$, if $\min(\Phi_1(w'), \Phi_2(w')) \not\geq v$ then $\Phi_1(w') \not\geq v$ or $\Phi_2(w') \not\geq v$. We assume without loss of generality that $\Phi_1(w) \not\geq \Phi(w)$. By Theorem 5.4.12, $\Phi_1(w) \not\geq \Phi(w)$ implies that for all $u' \prec w$ there exists $u' \preceq u'' \prec w$ such that $\pi_1(u'') \not\geq \Phi(w)$. Dually, since $\Phi(w) \geq \Phi(w)$, there exists $t \prec w$ such that $\pi(t') \geq \Phi(w)$ for all $t \preceq t' \prec w$. In particular, there exists $t \preceq t'' \prec w$ such that $\pi_1(t'') \not\geq \Phi(w)$ and $\pi(t'') \geq \Phi(w)$. By the definition of $\min$, we have that $\pi_1(t'') \geq \pi(t'') \geq \Phi(w)$ which contradicts that $\pi_1(t'') \not\geq \Phi(w)$. Hence, we proved that $\min(\Phi_1(w), \Phi_2(w)) \geq \Phi(w)$ for all $w \in \Sigma^\omega$.

Suppose towards contradiction that $\Phi(w) \not\geq \min(\Phi_1(w), \Phi_2(w))$ for some $w \in \Sigma^\omega$. In particular, $\mathsf{LimInf}_{u \prec w} \min(\pi_1(u), \pi_2(u)) \not\geq \min(\Phi_1(w), \Phi_2(w))$. Observe that for all $u \in \Sigma^*$ and $v \in \mathbb{D}$, if $\min(\pi_1(u), \pi_2(u)) \not\geq v$ then $\pi_1(u) \not\geq v$ or $\pi_2(u) \not\geq v$. We assume without loss of generality that $|\{u \mid \exists u \preceq u' \prec w : \pi_1(u') \not\geq \min(\Phi_1(w), \Phi_2(w))\}| = \infty$, or equivalently for all $u \prec w$, there exists $u \preceq u' \prec w$ such that $\pi_1(u') \not\geq \min(\Phi_1(w), \Phi_2(w))$. By Theorem 5.4.12, we get $\Phi_1(w) \not\geq \min(\Phi_1(w), \Phi_2(w))$. By the definition of $\min$, we have that $\Phi_1(w) \geq \min(\Phi_1(w), \Phi_2(w))$ which contradicts that $\Phi_1(w) \not\geq \min(\Phi_1(w), \Phi_2(w))$. Hence, we proved that $\Phi(w) \geq \min(\Phi_1(w), \Phi_2(w))$ for all $w \in \Sigma^\omega$. □

Now, we show that the tail-minimal response-time property can be expressed as a countable supremum of Inf-properties.

Example 5.4.14. *Let $i \in \mathbb{N}$ and define $\pi_{i,\mathsf{last}}$ as a finitary property that imitates π_{last} from Theorem 5.3.2, but ignores the first i observations of every finite trace. Formally, for all $u \in \Sigma^*$, we define $\pi_{i,\mathsf{last}}(u) = \pi_{\mathsf{last}}(u')$ if $u = u_i u'$ where $u_i \preceq u$ with $|u_i| = i$ and $u' \in \Sigma^*$, and $\pi_{i,\mathsf{last}}(u) = \infty$ otherwise. Observe that an equivalent way to define Φ_{tmin} from Theorem 5.4.11 is $\sup_{i \in \mathbb{N}}(\mathsf{Inf}_{u \prec w}(\pi_{i,\mathsf{last}}(u)))$ for all $w \in \Sigma^\omega$. Intuitively, for each $i \in \mathbb{N}$, we obtain an* Inf*-property that computes the minimal response time of the suffixes of a given trace. Taking the supremum over these, we obtain the greatest lower bound on all but finitely many response times.*

We generalize this observation and show that every LimInf-property is a countable supremum of Inf-properties.

Theorem 5.4.15. *Every* LimInf*-property is a countable supremum of* Inf*-properties.*

Proof. Let $\Phi = (\pi, \mathsf{LimInf})$. For each $i \in \mathbb{N}$ let us define $\Phi_i = (\pi_i, \mathsf{Inf})$ where π_i is as follows: $\pi_i(u) = \top$ if $|u| < i$, and $\pi_i(u) = \pi(u)$ otherwise. We claim that $\Phi(w) = \sup_{i\in\mathbb{N}} \Phi_i(w)$ for all $w \in \Sigma^\omega$. Expanding the definitions, observe that the claim is $\mathsf{LimInf}_{u\prec w}\pi(u) = \sup_{i\in\mathbb{N}} \mathsf{Inf}_{u\prec w}\pi_i(u)$. Due to the definition of LimInf, the expression $\sup_{i\in\mathbb{N}} \mathsf{Inf}_{u\prec w\wedge|u|\geq i}\pi(u)$ equals the left-hand side. Moreover, by the definition of π_i, it equals the right-hand side. □

We would also like to have the converse of Theorem 5.4.15, i.e., that every countable supremum of Inf-properties is a LimInf-property. Currently, we are able to show only the following.

Proposition 5.4.16. *Consider an infinite sequence $(\Phi_i)_{i\in\mathbb{N}}$ of properties with $\Phi_i = (\pi_i, \mathsf{Inf})$ for each $i \in \mathbb{N}$. The property $\Phi = (\pi, \mathsf{LimInf})$ where $\pi(u) = \max_{i\leq|u|} \pi_i(u)$ for all $u \in \Sigma^*$ satisfies $\sup_{i\in\mathbb{N}} \Phi_i(w) \leq \Phi(w)$ for all $w \in \Sigma^\omega$.*

Proof. For each $i \in \mathbb{N}$, assume without loss of generality that each π_i is nonincreasing. Let $\Phi = (\pi, \mathsf{LimInf})$ be as in the statement. We want to show that $\sup_{i\in\mathbb{N}} \Phi_i(w) \leq \Phi(w)$ for all $w \in \Sigma^\omega$. Expanding the definitions, observe that the claim is the following: $\sup_{i\in\mathbb{N}}(\mathsf{Inf}_{u\prec w}\pi_i(u)) \leq \mathsf{LimInf}_{u\prec w}(\max_{i\leq|u|} \pi_i(u))$ for all $w \in \Sigma^\omega$.

Let $w \in \Sigma^\omega$, and for each $k \in \mathbb{N}$, let $n_k = \max_{i\leq k} \mathsf{Inf}_{u\prec w}\pi_i(u)$ and $m_k = \max_{i\leq k} \pi_i(u_k)$ where $u_k \prec w$ with $|u_k| = k$. Observe that we have $n_k \leq m_k$ for all $k \in \mathbb{N}$. Then, we have $\liminf_{k\to\infty} n_k \leq \liminf_{k\to\infty} m_k$. Moreover, since the sequence $(n_k)_{k\in\mathbb{N}}$ is nondecreasing, we can replace the $\liminf$ on the left-hand side with $\lim$ to obtain the following: $\lim_{k\to\infty} \max_{i\leq k} \mathsf{Inf}_{u\prec w}\pi_i(u) \leq \liminf_{k\to\infty} \max_{i\leq k} \pi_i(u_k)$. Then, rewriting the expression concludes the proof by giving us $\sup_{i\in\mathbb{N}}(\mathsf{Inf}_{u\prec w}\pi_i(u)) \leq \mathsf{LimInf}_{u\prec w}(\max_{i\leq|u|} \pi_i(u))$. □

Remark 5.4.17. *Consider an infinite sequence $(\Phi_i)_{i\in\mathbb{N}}$ of finitely-converging* Inf*-properties, i.e., for every $i \in \mathbb{N}$ and every infinite word w there is a prefix $u \prec w$ such that $\Phi_i(w) = \Phi(uw')$ for all continuations w'. Evidently, each Φ_i is also a* Sup *property. Moreover, since* Sup*-properties are closed under countable suprema, $\sup_{i\in\mathbb{N}} \Phi_i$ is a* Sup*-property, and thus a* LimInf*-property by Theorem 5.4.10.*

We conjecture that some LimInf-property that is an upper bound like in Theorem 5.4.16 is also a lower bound on the countable supremum that occurs in the theorem. (The property Φ in Theorem 5.4.16 is not one.) This, together with Theorem 5.4.16, would imply the converse of Theorem 5.4.15. Proving the converse of Theorem 5.4.15 would give us, thanks to the following duality, that the LimInf- and LimSup-properties respectively characterize the countable suprema of Inf-properties and countable infima of Sup-properties, completing the picture for the generalization of the safety-progress hierarchy to the quantitative setting.

Proposition 5.4.18. *A property Φ is a* LimInf*-property iff its complement $\overline{\Phi}$ is a* LimSup*-property.*

5.5 Approximate Monitoring through Approximate Safety

In this section, we consider properties on extended reals $\mathbb{R}^{\pm\infty} = \mathbb{R} \cup \{-\infty, +\infty\}$. We denote by $\mathbb{R}_{\geq 0}$ the set of nonnegative real numbers.

Definition 5.5.1 (Approximate safety and co-safety)**.** *Let $\alpha \in \mathbb{R}_{\geq 0}$. A property Φ is* α-safe *iff for every $w \in \Sigma^\omega$ and value $v \in \mathbb{R}^{\pm\infty}$ with $\Phi(w) < v$, there exists a prefix $u \prec w$ such that $\sup_{w' \in \Sigma^\omega} \Phi(uw') < v + \alpha$. Similarly, Φ is* α-co-safe *iff for every $w \in \Sigma^\omega$ and $v \in \mathbb{R}^{\pm\infty}$ with $\Phi(w) > v$, there exists $u \prec w$ such that $\inf_{w' \in \Sigma^\omega} \Phi(uw') > v - \alpha$. When Φ is α-safe (resp. α-co-safe) for some $\alpha \in \mathbb{R}_{\geq 0}$, we say that Φ is* approximately safe *(resp.* approximately co-safe*).*

Approximate safety can be characterized through the following relation with the safety closure.

Proposition 5.5.2. *Let $\alpha \in \mathbb{R}_{\geq 0}$. A property Φ is α-safe iff $\Phi^*(w) - \Phi(w) \leq \alpha$ for all $w \in \Sigma^\omega$.*

Proof. Let Φ and α be as above. We show each direction separately by contradiction. First, assume Φ is α-safe. Suppose towards contradiction that $\Phi^*(w) - \Phi(w) > \alpha$ for some $w \in \Sigma^\omega$. Let $v = \Phi^*(w) - \alpha$ and notice that since Φ is α-safe, there exists $u \prec w$ such that $\sup_{w' \in \Sigma^\omega} \Phi(uw') < v + \alpha = \Phi^*(w)$. By definition, we get $\sup_{w' \in \Sigma^\omega} \Phi(uw') < \inf_{t \prec w} \sup_{w' \in \Sigma^\omega} \Phi(tw')$, which is a contradiction.

Now, assume $\Phi^*(w) - \Phi(w) \leq \alpha$ for all $w \in \Sigma^\omega$. Suppose towards contradiction that Φ is not α-safe, i.e., there exists $w \in \Sigma^\omega$ and $v \in \mathbb{D}$ such that (i) $\Phi(w) < v$ and (ii) $\sup_{w' \in \Sigma^\omega} \Phi(uw') \geq v + \alpha$ for all $u \prec w$. Note that (i) implies $v + \alpha > \Phi(w) + \alpha$, and (ii) implies $\inf_{u \prec w} \sup_{w' \in \Sigma^\omega} \Phi(uw') \geq v + \alpha$. Combining the two with the definition of Φ^* we get $\Phi^*(w) > \Phi(w) + \alpha$, which is a contradiction. □

An analogue of Theorem 5.5.2 holds for approximate co-safety and the co-safety closure. Moreover, approximate safety and approximate co-safety are dual notions that are connected by the complement operation, similarly to their precise counterparts (Theorem 5.3.10).

5.5.1 The Intersection of Approximate Safety and Co-safety

Recall the notion of ghost monitors from the introduction. If, after a finite number of observations, all the possible prediction values are close enough, then we can simply freeze the current value and achieve a sufficiently small error. This happens for properties that are both approximately safe and approximately co-safe, generalizing the unfolding approximation of discounted properties [BH14] and Theorem 4.3.9.

Proposition 5.5.3. *For every limit property Φ and all $\alpha, \beta \in \mathbb{R}_{\geq 0}$, if Φ is α-safe and β-co-safe, then the set $S_\delta = \{u \in \Sigma^* \mid \sup_{t_1 \in \Sigma^*} \Phi(ut_1) - \inf_{t_2 \in \Sigma^*} \Phi(ut_2) \geq \delta\}$ is finite for all $\delta > \alpha + \beta$.*

Proof. Let $\alpha, \beta \in \mathbb{R}_{\geq 0}$ and Φ be a limit property that is α-safe and β-co-safe. Assume towards contradiction that $|S_\delta| = \infty$ for some $\delta > \alpha + \beta$. Notice that S_δ is prefix closed, i.e., for all $u, t \in \Sigma^*$ having both $t \preceq u$ and $u \in S_\delta$ implies $t \in S_\delta$. Then, by König's lemma, there exists $w \in \Sigma^\omega$ such that $u \in S_\delta$ for every prefix $u \prec w$. Let $u_i \prec w$ be the prefix of length i. We have that $\lim_{n \to \infty} (\sup_{t_1 \in \Sigma^*} \Phi(u_n t_1) - \inf_{t_2 \in \Sigma^*} \Phi(u_n t_2)) \geq \delta > \alpha + \beta$. This implies that $\Phi^*(w) - \Phi_*(w) > \alpha + \beta$, which contradicts the assumption that Φ is α-safe and β-co-safe. Hence S_δ is finite for all $\delta > \alpha + \beta$. □

Based on this proposition, we show that, for limit properties that are both approximately safe and approximately co-safe, the influence of the suffix on the property value is eventually negligible.

Theorem 5.5.4. *For every limit property Φ such that $\Phi(w) \in \mathbb{R}$ for all $w \in \Sigma^\omega$, and for all $\alpha, \beta \in \mathbb{R}_{\geq 0}$, if Φ is α-safe and β-co-safe, then for every $\delta > \alpha + \beta$ and every $w \in \Sigma^\omega$, there is a prefix $u \prec w$ such that for all continuations $w' \in \Sigma^* \cup \Sigma^\omega$, we have $|\Phi(uw') - \Phi(u)| < \delta$.*

Proof. Given $\alpha, \beta \in \mathbb{R}_{\geq 0}$ and Φ as in the statement, assume Φ is α-safe and β-co-safe. Let $\delta > \alpha + \beta$ and $w \in \Sigma^\omega$ be arbitrary. Let S_δ be as in Proposition 5.5.3. Since S_δ is finite and prefix closed, there exists $u \prec w$ such that $ut \notin S_\delta$ for all $t \in \Sigma^*$. Let $u \prec w$ be the shortest such prefix. By construction, $\sup_{t_1 \in \Sigma^*} \Phi(ut_1) - \inf_{t_2 \in \Sigma^*} \Phi(ut_2) < \delta$. Furthermore, for all $t \in \Sigma^*$, we trivially have $\inf_{t_2 \in \Sigma^*} \Phi(ut_2) \leq \Phi(ut) \leq \sup_{t_1 \in \Sigma^*} \Phi(ut_1)$. In particular, $\inf_{t_2 \in \Sigma^*} \Phi(ut_2) \leq \Phi(u) \leq \sup_{t_1 \in \Sigma^*} \Phi(ut_1)$ holds simply by taking $t = \varepsilon$. Then, one can easily obtain $-\delta < \Phi(ut) - \Phi(u) < \delta$ for all $t \in \Sigma^*$. Since Φ is a limit property, this implies $-\delta < \Phi(uw') - \Phi(u) < \delta$ for all $w' \in \Sigma^\omega$ as well. □

We illustrate this theorem with a *discounted safety* property.

Example 5.5.5. *Let $P \subseteq \Sigma^\omega$ be a boolean safety property. We define the finitary property $\pi_P : \Sigma^* \to [0, 1]$ as follows: $\pi_P(u) = 1$ if $uw \in P$ for some $w \in \Sigma^\omega$, and $\pi_P(u) = 1 - 2^{-|t|}$ otherwise, where $t \preceq u$ is the shortest prefix with $tw \notin P$ for all $w \in \Sigma^\omega$. The limit property $\Phi = (\pi_P, \mathsf{Inf})$ is called* discounted safety*. Because Φ is an $\inf$-property, it is safe by Theorem 5.4.5. Now consider the finitary property π'_P defined by $\pi'_P(u) = 1 - 2^{-|u|}$ if $uw \in P$ for some $w \in \Sigma^\omega$, and $\pi'_P(u) = 1 - 2^{-|t|}$ otherwise, where $t \preceq u$ is the shortest prefix with $tw \notin P$ for all $w \in \Sigma^\omega$. Let $\Phi' = (\pi'_P, \mathsf{Sup})$, and note that $\Phi(w) = \Phi'(w)$ for all $w \in \Sigma^\omega$. Hence Φ is also co-safe, because it is a* Sup*-property.*

Let $w \in \Sigma^\omega$ and $\delta > 0$. For every prefix $u \prec w$, the set of possible prediction values is either a subset of the range $[1 - 2^{-|u|}, 1]$ or the singleton $\{1 - 2^{-|t|}\}$, where $t \preceq u$ is chosen as above. In the latter case, we have $|\Phi(uw') - \Phi(u)| = 0 < \delta$ for all $w' \in \Sigma^* \cup \Sigma^\omega$. In the former case, since the range becomes smaller as the prefix grows, there is a prefix $u' \prec w$ with $2^{-|u'|} < \delta$, which yields $|\Phi(u'w') - \Phi(u')| < \delta$ for all $w' \in \Sigma^* \cup \Sigma^\omega$.

5.5.2 Finite-state Approximate Monitoring

Monitors with finite state spaces are particularly desirable, because finite automata enjoy a plethora of desirable closure and decidability properties. Here, we prove that properties that are both approximately safe and approximately co-safe can be monitored approximately by a finite-state monitor. First, we recall the notion of abstract (quantitative) monitor from Chapter 4.

A binary relation $\sim$ over Σ^* is an *equivalence relation* iff it is reflexive, symmetric, and transitive. Such a relation is *right-monotonic* iff $u_1 \sim u_2$ implies $u_1 t \sim u_2 t$ for all $u_1, u_2, t \in \Sigma^*$. For an equivalence relation $\sim$ over Σ^* and a finite trace $u \in \Sigma^*$, we write $[u]_\sim$ for the equivalence class of $\sim$ to which u belongs. When $\sim$ is clear from the context, we write $[u]$ instead. We denote by $\Sigma^*/\!\sim$ the quotient of the relation $\sim$.

An *abstract monitor* $\mathcal{M} = (\sim, \gamma)$ is a pair consisting of a right-monotonic equivalence relation $\sim$ on Σ^* and a function $\gamma\colon (\Sigma^*/\!\sim) \to \mathbb{R}$. The monitor $\mathcal{M}$ is *finite-state* iff the relation $\sim$ has finitely many equivalence classes.

Let $\delta_{\mathsf{fin}}, \delta_{\lim} \in \mathbb{R}$ be error bounds. We say that $\mathcal{M}$ is a $(\delta_{\mathsf{fin}}, \delta_{\lim})$-*monitor* for a given limit property $\Phi = (\pi, \mathsf{Val})$ iff for all $u \in \Sigma^*$ and all $w \in \Sigma^\omega$, we have $|\pi(u) - \gamma([u])| \leq \delta_{\mathsf{fin}}$ and $|\mathsf{Val}_{t \prec w}(\pi(t)) - \mathsf{Val}_{t \prec w}(\gamma([t]))| \leq \delta_{\lim}$.

Building on Theorem 5.5.4, we identify a sufficient condition to guarantee the existence of an abstract monitor with finitely many equivalence classes.

Theorem 5.5.6. *For every limit property Φ such that $\Phi(w) \in \mathbb{R}$ for all $w \in \Sigma^\omega$, and for all error bounds $\alpha, \beta \in \mathbb{R}_{\geq 0}$, if Φ is α-safe and β-co-safe, then for every real $\delta > \alpha + \beta$, there exists a finite-state (δ, δ)-monitor for Φ.*

Proof. Let $\alpha, \beta \in \mathbb{R}_{\geq 0}$, and Φ be a limit property such that $\Phi(w) \in \mathbb{R}$ for all $w \in \Sigma^\omega$. Assume Φ is α-safe and β-co-safe, and let $\delta > \alpha + \beta$. We show how to construct a finite-state (δ, δ)-monitor for Φ.

Consider the finite set S_δ from Proposition 5.5.3. If S_δ is empty, then $|\Phi(u_1) - \Phi(u_2)| \leq \delta$ holds for all $u_1, u_2 \in \Sigma^*$, and thus we can construct a trivial (δ, δ)-monitor for Φ simply by (arbitrarily) mapping all finite traces to $\Phi(\varepsilon)$. So, we assume without loss of generality that S_δ is not empty.

Consider the function $\preceq_{S_\delta} \colon \Sigma^* \to \Sigma^*$ such that $\preceq_{S_\delta}(u) = u$ if $u \in S_\delta$, and $\preceq_{S_\delta}(u) = u'$ otherwise, where $u' \preceq u$ is the the shortest prefix with $u' \notin S_\delta$. We let $\mathcal{M} = (\sim, \gamma)$ where $\sim \; = \{(u_1, u_2) \mid \preceq_{S_\delta}(u_1) = \preceq_{S_\delta}(u_2)\}$ and $\gamma([u]) = \Phi(\preceq_{S_\delta}(u))$. By construction, $\sim$ is right-monotonic and has at most $|\Sigma| \times |S_\delta|$ equivalence classes.

Now, we prove that $|\Phi(u) - \gamma([u])| \leq \delta$ for all $u \in \Sigma^*$. If $u \in S_\delta$, then $\gamma([u]) = \Phi(u)$ by definition, and the statement holds trivially. Otherwise, if $u \notin S_\delta$, we let $u' = \preceq_{S_\delta}(u)$, which gives us $|\Phi(u't_1) - \Phi(u't_2)| < \delta$ for all $t_1, t_2 \in \Sigma^*$. In particular, $|\Phi(u) - \gamma([u])| < \delta$ since $u' \preceq u$. We remark that since Φ is a limit property, an error of at most δ on finite traces implies an error of at most δ on infinite traces.

Finally, we prove that $\sim$ is right-monotonic. Let $u_1, u_2 \in \Sigma^*$ such that $u_1 \sim u_2$. Note that $u_1 \sim u_2$ implies $u_1 \in S_\delta \Leftrightarrow u_2 \in S_\delta$ by definition of $\preceq_{S_\delta}$. If $u_1, u_2 \in S_\delta$, then $\preceq_{S_\delta}$ is the identity function, and thus $u_1 t \sim u_2 t$ for all $t \in \Sigma^*$ trivially. Otherwise, if $u_1, u_2 \notin S_\delta$, we define $u = \preceq_{S_\delta}(u_1) = \preceq_{S_\delta}(u_2) \notin S_\delta$. By definition of $\preceq_{S_\delta}$, we have that $\preceq_{S_\delta}(u) \notin S_\delta$ implies $\preceq_{S_\delta}(ut) = \preceq_{S_\delta}(u)$ for all $t \in \Sigma^*$. In particular, $u_1 t \sim u_2 t$ for all $t \in \Sigma^*$. □

Due to Theorem 5.5.6, the discounted safety property of Example 5.5.5 has a finite-state monitor for every positive error bound. We remark that Theorem 5.5.6 is proved by a construction that generalizes the approach for the approximate determinization of discounted automata [BH14], which unfolds an automaton until the distance constraint is satisfied.

5.6 Quantitative Liveness

A boolean property $P \subseteq \Sigma^\omega$ is live in the boolean sense iff for every $u \in \Sigma^*$ there exists $w \in \Sigma^\omega$ with $uw \in P$ [AS85], in other words, a wrong membership hypothesis can never be dismissed by a finite prefix. Similarly as for safety, we take the perspective of the quantitative membership problem to define liveness: a property Φ is live iff whenever a property value is less than $\top$, there exists a value v for which the wrong hypothesis $\Phi(w) \geq v$ can never be dismissed by any finite witness $u \prec w$.

Definition 5.6.1 (Liveness)**.** *A property $\Phi : \Sigma^\omega \to \mathbb{D}$ is* live in $\mathbb{D}$ *when for all $w \in \Sigma^\omega$, if $\Phi(w) < \top$, then there exists a value $v \in \mathbb{D}$ such that $\Phi(w) \not\geq v$ and for all prefixes $u \prec w$, we have $\sup_{w' \in \Sigma^\omega} \Phi(uw') \geq v$.*

When we write that a property $\Phi : \Sigma^\omega \to \mathbb{D}$ is *live* (instead of live in $\mathbb{D}$), we mean that Φ is live in the value domain $\mathbb{D}_\Phi = \{v \in \mathbb{D} \mid v \leq \top_\Phi\}$, and we let $\top = \sup \mathbb{D}_\Phi$. This is motivated by the following remark showing that a property's liveness may be closely tied to its value domain.

Remark 5.6.2. *Liveness of a property may depend on the top value of its value domain. Consider a liveness property $\Phi : \Sigma^\omega \to \mathbb{D}$ and the value domains $\mathbb{D}_\Phi = \{v \in \mathbb{D} \mid v \leq \top_\Phi\}$ and $\mathbb{D}' = \mathbb{D} \cup \{\top'\}$ with $v < \top'$ for all $v \in \mathbb{D}$.*

The property $\Phi_1 : \Sigma^\omega \to \mathbb{D}_\Phi$ where $\Phi(w) = \Phi_1(w)$ for all $w \in \Sigma^\omega$ is also live in $\mathbb{D}_\Phi$. This is easy to see by definition. For words $w \in \Sigma^\omega$ with $\Phi(w) = \top_\Phi$, the property is vacuously live in $\mathbb{D}_\Phi$, and those with $\Phi(w) < \top_\Phi$, it is live in $\mathbb{D}_\Phi$ thanks to its liveness in $\mathbb{D}$.

In contrast, $\Phi_2 : \Sigma^\omega \to \mathbb{D}'$ where $\Phi(w) = \Phi_2(w)$ for all $w \in \Sigma^\omega$ may be not live in $\mathbb{D}'$. For example, consider $\Sigma = \{a, b\}$ and $\mathbb{D} = \{0, x, y, 1\}$ where $0 < x < 1$ and $0 < y < 1$ but x and y are incomparable. Let $\Phi(w) = x$ if $w \in \Sigma^* a^\omega$, let $\Phi(w) = y$ if $w \in \Sigma^* b^\omega$, and let $\Phi(w) = 1$ otherwise. The property Φ is live in $\mathbb{D}$ since $\top_\Phi = \sup \mathbb{D} = 1$ and $\sup_{w \in \Sigma^\omega} \Phi(uw) = 1$ for every $u \in \Sigma^*$. However, considering the domain $\mathbb{D}' = \mathbb{D} \cup \{2\}$ with $1 < 2$, the same property is not live in $\mathbb{D}'$ because $\Phi((ab)^\omega) = 1$ and the only wrong lower bound hypothesis for $(ab)^\omega$ is $\sup \mathbb{D}' = 2$, which can be dismissed as $\sup_{w \in \Sigma^\omega} \Phi(uw) = 1$ for every prefix $u \prec (ab)^\omega$. In fact, for every property $\Phi : \Sigma^\omega \to \mathbb{D}$, if $\top_\Phi < \top$ and $\top_\Phi$ is attainable by some word, then Φ is not live in $\mathbb{D}$.

Let us first show that our definition of liveness generalizes the boolean one.

Proposition 5.6.3. *Quantitative liveness generalizes boolean liveness. In particular, for every boolean property $P \subseteq \Sigma^\omega$, the following statements are equivalent:*

1. *P is live according to the classical definition [AS85].*
2. *The characteristic property Φ_P is live in $\mathbb{B}$.*

Proof. Recall that (1) means the following: for every $w \notin P$ and every $u \prec w$ there exists $w' \in \Sigma^\omega$ such that $uw' \in P$. Expressing the same statement with the characteristic property Φ_P of P gives us the following: for every $w \in \Sigma^\omega$ if $\Phi_P(w) < 1$ then for every $u \prec w$ there exists $w' \in \Sigma^\omega$ such that $\Phi_P(uw') = 1$. Since $\mathbb{B} = \{0, 1\}$ and $0 < 1$, it is easy to see that this statement is equivalent to the definition of liveness in $\mathbb{B}$. □

Next, we provide a characterization of liveness through the safety closure operation.

Theorem 5.6.4. *A property Φ is live iff $SafetyCl(\Phi)(w) > \Phi(w)$ for every $w \in \Sigma^\omega$ with $\Phi(w) < \top$.*

Proof. First, suppose Φ is live. Let $w \in \Sigma^\omega$ be such that $\Phi(w) < \top$, and let v be as in the definition of liveness. Since $\sup_{w' \in \Sigma^\omega} \Phi(uw') \geq v$ for all prefixes $u \prec w$, we have that $SafetyCl(\Phi)(w) \geq v$. Moreover, since $v \not\leq \Phi(w)$, we are done. Now, suppose $SafetyCl(\Phi)(w) > \Phi(w)$ for every $w \in \Sigma^\omega$ with $\Phi(w) < \top$. Let $w \in \Sigma^\omega$ be such a trace, and let $v = SafetyCl(\Phi)(w)$. It is easy to see that v satisfies the liveness condition since $SafetyCl(\Phi)(w) = \inf_{u \prec w} \sup_{w' \in \Sigma^\omega} \Phi(uw') > \Phi(w)$. □

We show that liveness properties are closed under pairwise $\max$ considering totally-ordered value domains.

Proposition 5.6.5. *For every totally-ordered value domain $\mathbb{D}$, the set of liveness properties over $\mathbb{D}$ is closed under* $\max$.

Proof. Consider two properties $\Phi_1, \Phi_2 : \Sigma^\omega \to \mathbb{D}$ that are live in $\mathbb{D}$. Let Φ be their pairwise maximum, i.e., $\Phi(w) = \max(\Phi_1(w), \Phi_2(w))$ for all $w \in \Sigma^\omega$. We show that Φ fulfills the liveness definition for all $w \in \Sigma^\omega$. If $\Phi_1(w) = \top$ or $\Phi_2(w) = \top$ then $\Phi(w) = \top$. Otherwise, for each $i \in \{1, 2\}$, there exists v_i such that $\Phi_i(w) < v_i$, and for all $u \prec w$ we have $\sup_{w' \in \Sigma^\omega} \Phi_i(uw') \geq v_i$. Hence, because $\mathbb{D}$ is totally-ordered, defining $v = \max(v_1, v_2)$ implies $\Phi(w) < v$ as well as $\sup_{w' \in \Sigma^\omega} \Phi(uw') \geq v$ for all $u \prec w$. □

As in the boolean setting, the intersection of safety and liveness contains only the degenerate properties that are constant, i.e., always output $\top$.

Proposition 5.6.6. *A property Φ is safe and live iff $\Phi(w) = \top$ for all $w \in \Sigma^\omega$.*

Proof. Observe that the constant function $\Phi = \top$ is trivially safe and live. Now, let Φ be a property that is both safe and live, and suppose towards contradiction that $\Phi(w) < \top$ for some $w \in \Sigma^\omega$. Since Φ is live, there exists a value v with $\Phi(w) \not\geq v$ such that for all $u \prec w$, we have $\sup_{w' \in \Sigma^\omega} \Phi(uw') \geq v$. In particular, $\inf_{u \prec w} \sup_{w' \in \Sigma^\omega} \Phi(uw') \geq v$ and $\Phi(w) \not\geq v$ hold, implying $\mathit{SafetyCl}(\Phi)(w) > \Phi(w)$ by definition of safety closure and Theorem 5.3.6. Then, again by Theorem 5.3.6, this contradicts the assumption that Φ is safe. □

We define co-liveness symmetrically, and note that the duals of the statements above also hold for co-liveness.

Definition 5.6.7 (Co-liveness)**.** *A property $\Phi : \Sigma^\omega \to \mathbb{D}$ is* co-live in $\mathbb{D}$ *when for all $w \in \Sigma^\omega$, if $\Phi(w) > \bot$, then there exists a value $v \in \mathbb{D}$ such that $\Phi(w) \not\leq v$ and for all prefixes $u \prec w$, we have* $\inf_{w' \in \Sigma^\omega} \Phi(uw') \leq v$.

Next, we present some examples of liveness and co-liveness properties. We start by showing that LimInf- and LimSup-properties can be live and co-live.

Example 5.6.8. *Let $\Sigma = \{a, b\}$ be an alphabet, and let $P = (\Sigma^* a)^\omega$ (infinitely many a's) and $Q = \Sigma^\omega \setminus P$ (finitely many a's) be boolean properties. Consider their characteristic properties Φ_P and Φ_Q. As we pointed out earlier, our definitions generalize their boolean counterparts, therefore Φ_P and Φ_Q are both live and co-live. Moreover, Φ_P is a* LimSup*-property: define $\pi_P(u) = 1$ if $u \in \Sigma^* a$, and $\pi_P(u) = 0$ otherwise, and observe that $\Phi_P(w) = \text{LimSup}_{u \prec w} \pi_P(u)$ for all $w \in \Sigma^\omega$. Similarly, Φ_Q is a* LimInf*-property.*

Now, we show that the maximal response-time property is live, and the minimal response time is co-live.

Example 5.6.9. *Recall the co-safety property $\Phi_{\max}$ of maximal response time from Theorem 5.3.11. Let $w \in \Sigma^\omega$ such that $\Phi_{\max}(w) < \infty$. We can extend every prefix $u \prec w$ with $w' = \mathsf{rq}\,\mathsf{tk}^\omega$, which gives us $\Phi_{\max}(uw') = \infty > \Phi(w)$. Equivalently, for every $w \in \Sigma^\omega$, we have $\mathit{SafetyCl}(\Phi_{\max})(w) = \infty > \Phi_{\max}(w)$. Hence $\Phi_{\max}$ is live and, analogously, the safety property $\Phi_{\min}$ from Theorem 5.3.2 is co-live.*

We next present the *average response-time* property and show that it is live and co-live.

Example 5.6.10. *Let $\Sigma = \{\mathtt{rq}, \mathtt{gr}, \mathtt{tk}, \mathtt{oo}\}$. For all $u \in \Sigma^*$, let $p(u) = 1$ if there is no pending $\mathtt{rq}$ in u, and $p(u) = 0$ otherwise. Define $\pi_{valid}(u) = |\{u' \preceq u \mid \exists u'' \in \Sigma^* : u' = u''\mathtt{rq} \wedge p(u'') = 1\}|$ as the number of valid requests in u, and define $\pi_{time}(u)$ as the total number of $\mathtt{tk}$ observations that occur after a valid $\mathtt{rq}$ and before the matching $\mathtt{gr}$. Then, $\Phi_{\mathsf{avg}} = (\pi_{\mathsf{avg}}, \mathsf{LimInf})$, where $\pi_{\mathsf{avg}}(u) = \frac{\pi_{time}(u)}{\pi_{valid}(u)}$ for all $u \in \Sigma^*$ with $\pi_{valid}(u) > 0$, and $\pi_{\mathsf{avg}}(u) = \infty$ otherwise. For example, $\pi_{\mathsf{avg}}(u) = \frac{3}{2}$ for $u = \mathtt{rq\,tk\,gr\,tk\,rq\,tk\,rq\,tk}$. Note that Φ_{avg} is a* LimInf*-property.*

The property Φ_{avg} is defined on the value domain $[0, \infty]$ and is both live and co-live. To see this, let $w \in \Sigma^\omega$ such that $0 < \Phi_{\mathsf{avg}}(w) < \infty$ and, for every prefix $u \prec w$, consider $w' = \mathtt{rq\,tk}^\omega$ and $w'' = \mathtt{gr}\,(\mathtt{rq\,gr})^\omega$. Since uw' has a pending request followed by infinitely many clock ticks, we have $\Phi_{\mathsf{avg}}(uw') = \infty$. Similarly, since uw'' eventually has all new requests immediately granted, we get $\Phi_{\mathsf{avg}}(uw'') = 0$.

Notice that for the average response-time property Φ_{avg} in the example above, we have $\Phi_{\mathsf{avg}}(w) = \Phi_{\mathsf{avg}}(uw)$ for every $u \in \Sigma^*$ and $w \in \Sigma^\omega$. Such properties are called *prefix independent*. Finally, we show that every prefix-independent property is both live and co-live.

Proposition 5.6.11. *Every prefix-independent property Φ is live and co-live.*

Proof. Consider a prefix-independent property Φ. We only show that Φ is live as its co-liveness can be proved similarly. Suppose towards contradiction that Φ is not live, and thus by Theorem 5.6.4 that $\Phi(w) = SafetyCl(\Phi)(w)$ for some $w \in \Sigma^\omega$ with $\Phi(w) < \top$. Let w be such a word, and consider two prefixes $u_1 \preceq u_2 \prec w$ such that $\sup_{w' \in \Sigma^\omega} \Phi(u_2 w') < \sup_{w' \in \Sigma^\omega} \Phi(u_1 w')$. Such prefixes exist because otherwise we have a contradiction to $\Phi(w) < \top$. Then, there exists $w'' \in \Sigma^\omega$ such that $\Phi(u_2 w'') < \Phi(u_1 w'')$. Since $u_1 \preceq u_2$, there is a finite word u_3 with $u_2 = u_1 \cdot u_3$. Notice that, since Φ is prefix independent, we have $\Phi(w'') = \Phi(u_1 w'') = \Phi(u_1 u_3 w'')$, which contradicts $\Phi(u_2 w'') < \Phi(u_1 w'')$. □

5.6.1 The Quantitative Safety-Liveness Decomposition

A celebrated theorem states that every boolean property can be expressed as an intersection of a safety property and a liveness property [AS85]. In this section, we prove an analogous result in the quantitative setting.

Example 5.6.12. *Let $\Sigma = \{\mathtt{rq}, \mathtt{gr}, \mathtt{tk}, \mathtt{oo}\}$. Recall the maximal response-time property $\Phi_{\max}$ from Theorem 5.3.11, and the average response-time property Φ_{avg} from Theorem 5.6.10. Let $n > 0$ be an integer and define a new property $\Phi : \Sigma^\omega \to [0, n]$ by $\Phi(w) = \Phi_{\mathsf{avg}}(w)$ if $\Phi_{\max}(w) \leq n$, and $\Phi(w) = 0$ otherwise. For the safety closure of Φ, we have $SafetyCl(\Phi)(w) = n$ if $\Phi_{\max}(w) \leq n$, and $SafetyCl(\Phi)(w) = 0$ otherwise. Now, we further define $\Psi(w) = \Phi_{\mathsf{avg}}(w)$ if $\Phi_{\max}(w) \leq n$, and $\Psi(w) = n$ otherwise. Observe that Ψ is live, because every prefix of a trace whose value is less than n can be extended to a greater value. Finally, note that for all $w \in \Sigma^\omega$, we can express $\Phi(w)$ as the pointwise minimum of $SafetyCl(\Phi)(w)$ and $\Psi(w)$. Intuitively, the safety part $SafetyCl(\Phi)$ of this decomposition checks whether the maximal response time stays below the permitted bound, and the liveness part Ψ keeps track of the average response time as long as the bound is satisfied.*

Following a similar construction, we show that a safety-liveness decomposition exists for every property.

Theorem 5.6.13. *For every property Φ, there exists a liveness property Ψ such that $\Phi(w) = \min(SafetyCl(\Phi)(w), \Psi(w))$ for all $w \in \Sigma^\omega$.*

Proof. Let Φ be a property and consider its safety closure $SafetyCl(\Phi)$. We define Ψ as follows: $\Psi(w) = \Phi(w)$ if $SafetyCl(\Phi)(w) \neq \Phi(w)$, and $\Psi(w) = \top$ otherwise. Note that $SafetyCl(\Phi)(w) \geq \Phi(w)$ for all $w \in \Sigma^\omega$ by Theorem 5.3.6. When $SafetyCl(\Phi)(w) > \Phi(w)$, we have $\min(SafetyCl(\Phi)(w), \Psi(w)) = \min(SafetyCl(\Phi)(w), \Phi(w)) = \Phi(w)$. When $SafetyCl(\Phi)(w) = \Phi(w)$, we have $\min(SafetyCl(\Phi)(w), \Psi(w)) = \min(\Phi(w), \top) = \Phi(w)$.

Now, suppose towards contradiction that Ψ is not live, i.e., there exists $w \in \Sigma^\omega$ such that $\Psi(w) < \top$ and for all $v \not\leq \Phi(w)$, there exists $u \prec w$ satisfying $\sup_{w' \in \Sigma^\omega} \Phi(uw') \not\geq v$. Let $w \in \Sigma^\omega$ be such that $\Psi(w) < \top$. Then, by definition of Ψ, we know that $\Psi(w) = \Phi(w) < SafetyCl(\Phi)(w)$. Moreover, since $SafetyCl(\Phi)(w) \not\leq \Psi(w)$, there exists $u \prec w$ satisfying $\sup_{w' \in \Sigma^\omega} \Phi(uw') \not\geq SafetyCl(\Phi)(w)$. In particular, we have $\sup_{w' \in \Sigma^\omega} \Phi(uw') < SafetyCl(\Phi)(w)$. Since we have $SafetyCl(\Phi)(w) = \inf_{u' \prec w} \sup_{w' \in \Sigma^\omega} \Phi(u'w')$ by definition and $u \prec w$, it yields a contradiction. Therefore, Ψ is live. □

In particular, if the given property is safe or live, the decomposition is trivial.

Remark 5.6.14. *Let Φ be a property. If Φ is safe, then the safety part of the decomposition is Φ itself, and the liveness part is the constant property that maps every trace to $\top$. If Φ is live, then the liveness part of the decomposition is Φ itself, and the safety part is $SafetyCl(\Phi)$. Note that, in this case, $SafetyCl(\Phi)$ may differ from the constant function $\top$, but taking the safety part as constant function $\top$ is a valid decomposition.*

Another decomposition theorem is the one of boolean properties over nonunary alphabets into two liveness properties [AS85]. We extend this result to the quantitative setting.

Theorem 5.6.15. *For every property Φ over a nonunary alphabet Σ, there exist two liveness properties Ψ_1 and Ψ_2 such that $\Phi(w) = \min(\Psi_1(w), \Psi_2(w))$ for all $w \in \Sigma^\omega$.*

Proof. Let Σ be a finite alphabet with $|\Sigma| \geq 2$ and $a_1, a_2 \in \Sigma$ be two distinct letters. Consider an arbitrary property Φ over Σ. For $i \in \{1, 2\}$, we define Ψ_i as follows: $\Psi_i(w) = \top$ if $w = u(a_i)^\omega$ for some $u \in \Sigma^*$, and $\Psi_i(w) = \Phi(w)$ otherwise. Note that, since a_1 and a_2 are distinct, whenever $w \in \Sigma^*(a_1)^\omega$ then $w \notin \Sigma^*(a_2)^\omega$, and vice versa. Then, we have that both Ψ_1 and Ψ_2 are $\top$ only when Φ is $\top$. In the remaining cases, when at most one of Ψ_1 and Ψ_2 is $\top$, then either both equals Φ or one of them is $\top$ and the other is Φ. As a direct consequence, $\Phi(w) = \min(\Psi_1(w), \Psi_2(w))$ for all $w \in \Sigma^\omega$.

Now, we show that Ψ_1 and Ψ_2 are both live. By construction, $\Psi_i(u(a_i)^\omega) = \top$ for all $u \in \Sigma^*$. In particular, $SafetyCl(\Psi_i)(w) = \inf_{u \prec w} \sup_{w' \in \Sigma^\omega} \Psi_i(uw') = \top$ for all $w \in \Sigma^\omega$. We conclude that Ψ_i is live thanks to Theorem 5.6.4. □

For co-safety and co-liveness, the duals of Theorem 5.6.14 and Theorems 5.6.13 and 5.6.15 hold. In particular, every property is the pointwise maximum of its co-safety closure and a co-liveness property.

5.6.2 Threshold Liveness and Top Liveness

Threshold liveness connects a quantitative property and the boolean liveness of the sets of words whose values exceed a threshold value.

Definition 5.6.16 (Threshold liveness and co-liveness)**.** *A property $\Phi : \Sigma^\omega \to \mathbb{D}$ is* threshold live *when for every $v \in \mathbb{D}$ the boolean property $\Phi_{\geq v}$ is live (and thus $\Phi_{\not\geq v}$ is co-live). Equivalently, Φ is threshold live when for every $u \in \Sigma^*$ and $v \in \mathbb{D}$ there exists $w \in \Sigma^\omega$ such that $\Phi(uw) \geq v$. Similarly, a property $\Phi : \Sigma^\omega \to \mathbb{D}$ is* threshold co-live *when for every $v \in \mathbb{D}$ the boolean property $\Phi_{\not\leq v}$ is co-live (and thus $\Phi_{\leq v}$ is live). Equivalently, Φ is threshold co-live when for every $u \in \Sigma^*$ and $v \in \mathbb{D}$ there exists $w \in \Sigma^\omega$ such that $\Phi(uw) \leq v$.*

We relate threshold liveness with the boolean liveness of a single set of words.

Proposition 5.6.17. *A property Φ is threshold live iff the set $\Phi_{\geq \top}$ is live in the boolean sense.*

Proof. Consider a property $\Phi : \Sigma^\omega \to \mathbb{D}$.

($\Rightarrow$): Assume Φ to be threshold live, i.e., for every $v \in \mathbb{D}$ the boolean property $\Phi_{\geq v}$ is live. In particular, $\Phi_{\geq \top}$ is also live.

($\Leftarrow$): Assume $\Phi_{\geq \top}$ to be live in the boolean sense. Observe that for every $v \leq \top$ we have $\Phi_{\geq \top} \subseteq \Phi_{\geq v}$. Since the union of a boolean liveness property with any boolean property is live [AS85], the boolean property $\Phi_{\geq v}$ is also live for all $v \leq \top$, i.e., Φ is threshold live. □

Liveness is characterized by the safety closure being strictly greater than the property whenever possible (Theorem 5.6.4). Top liveness puts an additional requirement on liveness: the safety closure of the property should not only be greater than the original property but also equal to the top value.

Definition 5.6.18 (Top liveness and bottom co-liveness)**.** *A property Φ is* top live *when $SafetyCl(\Phi)(w) = \top$ for every $w \in \Sigma^\omega$. Similarly, a property Φ is* bottom co-live *when $CoSafetyCl(\Phi)(w) = \bot$ for every $w \in \Sigma^\omega$.*

We provide a strict hierarchy of threshold liveness, top liveness, and liveness.

Proposition 5.6.19. *Every threshold-live property is top live, but not vice versa; and every top-live property is live, but not vice versa.*

Proof. First, we show the strict inclusion of threshold liveness in top liveness. Let Φ be a threshold-live property. In particular, taking the threshold $v = \top$ gives us that for every $u \in \Sigma^*$ there exists $w \in \Sigma^\omega$ such that $\Phi(uw) = \top$. Then, $\sup_{w \in \Sigma^\omega} \Phi(uw) = \top$ for all $u \in \Sigma^*$, which implies that Φ is top live. Next, consider the property Φ over the alphabet $\{a, b\}$, defined for all $w \in \Sigma^\omega$ as follows: $\Phi(w) = |w|_a$ if w has finitely many a's, otherwise $\Phi(w) = 0$. Observe that $\sup_{w \in \Sigma^\omega} \Phi(uw) = \infty$ for all $u \in \Sigma^*$, therefore it is top live. However, for the threshold $v = \infty$, the set $\Phi_{\geq v}$ is empty, implying that it is not threshold live.

Now, we show that the strict inclusion of top liveness in liveness. Recall that, by Theorem 5.6.4, a property Φ is live iff for every $w \in \Sigma^\omega$ if $\Phi(w) < \top$ then $\Phi(w) < SafetyCl(\Phi)(w)$. Then, notice that if a property Φ is top live, it is obviously live. Next, let $\Sigma = \{a, b\}$ and consider the property $\Phi : \Sigma^\omega \to [0, 2]$ defined for all $w \in \Sigma^\omega$ as follows: $\Phi(w) = 0$ if w is of the form $\Sigma^* b^\omega$, otherwise $\Phi(w) = \sum_{i \geq 0} 2^{-i} f(\sigma_i)$ where $w = \sigma_0 \sigma_1 \ldots$, $f(a) = 0$, and $f(b) = 1$. Observe that

Φ is live since $\Phi(w) < \mathit{SafetyCl}(\Phi)(w)$ for every word $w \in \Sigma^\omega$. However, it is not top live since $\mathit{SafetyCl}(\Phi)(aw) \leq 1 < 2 = \top$ for all $w \in \Sigma^\omega$. □

Top liveness does not imply threshold liveness, but it does imply a weaker form of it.

Proposition 5.6.20. *For every top-live property Φ and value $v < \top$, the set $\Phi_{\geq v}$ is live in the boolean sense.*

Proof. Let Φ be top live property, i.e., $\inf_{u \prec w} \sup_{w' \in \Sigma^\omega} \Phi(uw') = \top$ for all $w \in \Sigma^\omega$. Let $v < \top$ be a value. Suppose towards contradiction that $\Phi_{\geq v}$ is not live in the boolean sense, i.e., there exists $\hat{u} \in \Sigma^*$ such that $\Phi(\hat{u}w') \not\geq v$ for all $w' \in \Sigma^\omega$. Let $\hat{w} \in \Sigma^\omega$ be such that $\hat{u} \prec \hat{w}$. Clearly $\inf_{u \prec \hat{w}} \sup_{w' \in \Sigma^\omega} \Phi(uw') \not\geq v$. Either $\inf_{u \prec \hat{w}} \sup_{w' \in \Sigma^\omega} \Phi(uw')$ is incomparable with v, or it is less than v. Since $\top$ compares with all values, we have that $\inf_{u \prec \hat{w}} \sup_{w' \in \Sigma^\omega} \Phi(uw') < \top$, which contradicts the top liveness of Φ. □

While the three liveness notions differ in general, they do coincide for $\sup$-closed properties.

Theorem 5.6.21. *A $\sup$-closed property is live iff it is top live iff it is threshold live.*

Proof. Notice that for every $\sup$-closed property Φ, top liveness means that for every $u \in \Sigma^*$ there is $w \in \Sigma^\omega$ such that $\Phi(uw) = \top$. Let Φ be a $\sup$-closed liveness property. Suppose towards contradiction that it is not top live, i.e., there is $u \in \Sigma^*$ such that for all $w \in \Sigma^\omega$ we have $\Phi(uw) < \top$. Let $\sup_{w \in \Sigma^\omega} \Phi(uw) = k < \top$, and note that since Φ is $\sup$-closed, there exists an infinite continuation $w' \in \Sigma^\omega$ for which $\Phi(uw') = k < \top$. As Φ is live, there exists a value v such that $k \not\geq v$ and for every prefix $u' \prec uw'$ there exists $w'' \in \Sigma^\omega$ with $\Phi(u'w'') \geq v$. However, letting $u' = u$ yields a contradiction to our initial supposition.

Now, let Φ be a $\sup$-closed top liveness property. Thanks to Theorem 5.6.17, it is sufficient to show that the boolean property $\Phi_{\geq \top}$ is live in the boolean sense. Suppose towards contradiction that $\Phi_{\geq \top}$ is not live, i.e., there exists $u \in \Sigma^*$ such that for all $w \in \Sigma^\omega$ we have $\Phi(uw) < \top$. Due to $\sup$-closedness, we have $\sup_{w \in \Sigma^\omega} \Phi(uw) < \top$ as well. Moreover, for every $w \in \Sigma^\omega$ such that $u \prec w$, this means that $\inf_{u \prec w} \sup_{w' \in \Sigma^\omega} \Phi(uw') < \top$, which is a contradiction. □

5.6.3 Additional Notions Related to Quantitative Liveness

In [LDL17], the authors define a property Φ as *multi-live* iff $\mathit{SafetyCl}(\Phi)(w) > \bot$ for all $w \in \Sigma^\omega$. We show that our definition is more restrictive, resulting in fewer liveness properties while still allowing a safety-liveness decomposition.

Proposition 5.6.22. *Every live property is multi-live, but not vice versa.*

Proof. We prove that liveness implies multi-liveness. Suppose toward contradiction that some property Φ is live, but not multi-live. Then, there exists $w \in \Sigma^\omega$ for which $\mathit{SafetyCl}(\Phi)(w) = \bot$, and therefore $\Phi(w) = \bot$ too. Note that we assume $\mathbb{D}$ is a nontrivial complete lattice, i.e., $\top \neq \bot$. Then, since Φ is live, we have $\mathit{SafetyCl}(\Phi)(w) > \Phi(w)$ by Theorem 5.6.4, which yields a contradiction.

Now, we provide a separating example on a totally ordered domain. Let $\Sigma = \{a, b, c\}$, and consider the following property: $\Phi(w) = 0$ if $w = a^\omega$, and $\Phi(w) = 1$ if $w \in \Sigma^* c \Sigma^\omega$, and $\Phi(w) = 2$ otherwise (i.e., if w has some b and no c). For all $w \in \Sigma^\omega$ and prefixes

$u \prec w$, we have $\Phi(uc^\omega) = 1$. Thus $SafetyCl(\Phi)(w) \neq \bot$, which implies that Φ is multi-live. However, Φ is not live. Indeed, for every $w \in \Sigma^\omega$ such that $w \in \Sigma^* c \Sigma^\omega$, we have $\Phi(w) = 1 < \top$. Moreover, w admits some prefix u that contains an occurrence of c, thus satisfying $\sup_{w' \in \Sigma^\omega} \Phi(uw') = 1$. □

Recall that a property is both safe and live iff it is the constant function $\top$ (Theorem 5.6.6). For multi-safety and multi-liveness, this is not the case.

Example 5.6.23. *Let $\Sigma = \{a, b\}$ be an alphabet and $\mathbb{D} = \{v_1, v_2, \bot, \top\}$ be a lattice where v_1 and v_2 are incomparable. Consider the property $\Phi : \Sigma^\omega \to \mathbb{D}$ that is defined as $\Phi(w) = v_1$ if $a \prec w$ and $\Phi(w) = v_2$ if $b \prec w$. Recall from Theorem 5.3.12 that Φ is safe, thus multi-safe by Theorem 5.3.29. Clearly, $SafetyCl(\Phi)(w) > \bot$ for all $w \in \Sigma^\omega$, thus Φ is multi-live. However, Φ is not live as for all words w, we have $\Phi(w) = SafetyCl(\Phi)(w) < \top$.*

In [GS22], the authors define a property Φ as *verdict-live* iff for every $w \in \Sigma^\omega$ and value $v \not\leq \Phi(w)$, every prefix $u \prec w$ satisfies $\Phi(uw') = v$ for some $w' \in \Sigma^\omega$. We show that our definition is more liberal.

Proposition 5.6.24. *Every verdict-live property is live, but not vice versa.*

Proof. The implication holds trivially. We provide a separating example below, concluding that our definition is strictly more general even for totally ordered domains. Let $\Sigma = \{a, b\}$, and consider the following property: $\Phi(w) = 0$ if $w = a^\omega$, and $\Phi(w) = 1$ if $w \in \Sigma^* b \Sigma^* b \Sigma^\omega$, and $\Phi(w) = 2^{-|u|}$ otherwise (if w has exactly one b), where $u \prec w$ is the shortest prefix in which b occurs. Consider an arbitrary $w \in \Sigma^\omega$. If $\Phi(w) = 1$, then the liveness condition is vacuously satisfied. If $\Phi(w) = 0$, then $w = a^\omega$, and every prefix $u \prec w$ can be extended with $w' = ba^\omega$ or $w'' = b^\omega$ to obtain $\Phi(uw') = 2^{-(|u|+1)}$ and $\Phi(uw'') = 1$. If $0 < \Phi(w) < 1$, then w has exactly one b, and every prefix $u \prec w$ can be extended with b^ω to obtain $\Phi(ub^\omega) = 1$. Hence Φ is live. However, Φ is not verdict-live. To see this, consider the trace $w = a^k b a^\omega$ for some integer $k \geq 1$ and note that $\Phi(w) = 2^{-(k+1)}$. Although all prefixes of w can be extended to achieve the value 1, the value domain contains elements between $\Phi(w)$ and 1, namely the values 2^{-m} for $1 \leq m \leq k$. Each of these values can be rejected after reading a finite prefix of w, because for $n \geq m$ it is not possible to extend a^n to achieve 2^{-m}. □

Let us conclude with a remark on the form of hypotheses in our definition of liveness.

Remark 5.6.25. *In the same vein as Theorem 5.3.34, suppose we define liveness with strict lower bound hypotheses instead of nonstrict: for all $w \in \Sigma^\omega$, if $\Phi(w) < \top$, then there exists a value $v \in \mathbb{D}$ such that $\Phi(w) \not> v$ and for all prefixes $u \prec w$, we have $\sup_{w' \in \Sigma^\omega} \Phi(uw') > v$. Let w be a word with $\Phi(w) < \top$ and consider $v = \Phi(w)$. Evidently, according to this definition, it would be permissible for the $\sup$ of possible prediction values to converge to $\Phi(w)$, in other words, for the safety closure to have the same value as the property on a word whose value is less than $\top$, which is too lenient.*

5.7 Quantitative Automata

A *nondeterministic quantitative automaton* (or just automaton from here on) on words is a tuple $\mathcal{A} = (\Sigma, Q, \iota, \delta)$, where Σ is an alphabet; Q is a finite nonempty set of states; $\iota \in Q$ is an initial state; and $\delta \colon Q \times \Sigma \to 2^{(\mathbb{Q} \times Q)}$ is a finite transition function over weight-state pairs.

A *transition* is a tuple $(q, \sigma, x, q') \in Q \times \Sigma \times \mathbb{Q} \times Q$, such that $(x, q') \in \delta(q, \sigma)$, also written $q \xrightarrow{\sigma:x} q'$. We write $\gamma(t) = x$ for the weight of a transition $t = (q, \sigma, x, q')$. $\mathcal{A}$ is deterministic if for all $q \in Q$ and $\sigma \in \Sigma$, the set $\delta(q, \sigma)$ is a singleton. We require the automaton $\mathcal{A}$ to be *total*, namely that for every state $q \in Q$ and letter $\sigma \in \Sigma$, there is at least one state q' and a transition $q \xrightarrow{\sigma:x} q'$. For a state $q \in Q$, we denote by $\mathcal{A}^q$ the automaton that is derived from $\mathcal{A}$ by setting its initial state ι to q.

A run of $\mathcal{A}$ on a word w is a sequence $\rho = q_0 \xrightarrow{w[0]:x_0} q_1 \xrightarrow{w[1]:x_1} q_2 \dots$ of transitions where $q_0 = \iota$ and $(x_i, q_{i+1}) \in \delta(q_i, w[i])$. For $0 \leq i < |w|$, we denote the ith transition in ρ by $\rho[i]$, and the finite prefix of ρ up to and including the ith transition by $\rho[..i]$. As each transition t_i carries a weight $\gamma(t_i) \in \mathbb{Q}$, the sequence ρ provides a weight sequence $\gamma(\rho) = \gamma(t_0)\gamma(t_1)\dots$ A Val-automaton is one equipped with a value function $\mathsf{Val} : \mathbb{Q}^\omega \to \mathbb{R}$, which assigns real values to runs of $\mathcal{A}$.

We assume that Val is bounded for every finite set of rationals, i.e., for every finite $V \subset \mathbb{Q}$ there exist $m, M \in \mathbb{R}$ such that $m \leq \mathsf{Val}(x) \leq M$ for every $x \in V^\omega$. The finite set V corresponds to transition weights of a quantitative automaton, and the concrete value functions we consider satisfy this assumption.

Notice that while quantitative properties can be defined over arbitrary value domains, we restrict quantitative automata to totally-ordered numerical value domains (i.e., bounded subsets of $\mathbb{R}$) as this is the standard setting in the literature.

The value of a run ρ is $\mathsf{Val}(\gamma(\rho))$. The value of a Val-automaton $\mathcal{A}$ on a word w, denoted $\mathcal{A}(w)$, is the supremum of $\mathsf{Val}(\rho)$ over all runs ρ of $\mathcal{A}$ on w, generalizing the standard approach in boolean automata where acceptance is defined through the existence of an accepting run.

The *top value* of a Val-automaton $\mathcal{A}$ is $\top_{\mathcal{A}} = \sup_{w \in \Sigma^\omega} \mathcal{A}(w)$, which we denote by $\top$ when $\mathcal{A}$ is clear from the context. Note that when we speak of the top value of an automaton or a property expressed by an automaton, we always match its value domain to have the same top value.

An automaton $\mathcal{A}$ is *safe* (resp. *live*) iff it expresses a safety (resp. liveness) property. Two automata $\mathcal{A}$ and $\mathcal{A}'$ are *equivalent*, if they express the same function from words to reals. The size of an automaton consists of the maximum among the size of its alphabet, state-space, and transition-space, where weights are represented in binary.

We list below the value functions for quantitative automata that we will use, defined over infinite sequences $v_0 v_1 \dots$ of rational weights.

- $\mathsf{Inf}(v) = \inf\{v_n \mid n \geq 0\}$
- $\mathsf{Sup}(v) = \sup\{v_n \mid n \geq 0\}$
- $\mathsf{LimInf}(v) = \lim_{n\to\infty} \inf\{v_i \mid i \geq n\}$
- $\mathsf{LimSup}(v) = \lim_{n\to\infty} \sup\{v_i \mid i \geq n\}$
- $\mathsf{LimInfAvg}(v) = \mathsf{LimInf}\left(\frac{1}{n}\sum_{i=0}^{n-1} v_i\right)$
- $\mathsf{LimSupAvg}(v) = \mathsf{LimSup}\left(\frac{1}{n}\sum_{i=0}^{n-1} v_i\right)$
- For a discount factor $\lambda \in \mathbb{Q} \cap (0, 1)$, $\mathsf{DSum}_\lambda(v) = \sum_{i\geq 0} \lambda^i v_i$

A value function Val is *prefix independent* iff for all $x \in \mathbb{Q}^*$ and all $y \in \mathbb{Q}^\omega$ we have $\mathsf{Val}(y) = \mathsf{Val}(xy)$. The value functions LimInf, LimSup, LimInfAvg, and LimSupAvg are prefix independent, while Inf, Sup, and DSum are not.

The following statement allows us to consider Inf- and Sup-automata as only having runs with respectively nonincreasing and nondecreasing sequences of weights, and to also consider them as LimInf- and LimSup-automata.

Proposition 5.7.1. *Let* Val $\in \{\mathsf{Inf}, \mathsf{Sup}\}$. *Given a* Val*-automaton, we can construct in* PTIME *an equivalent* Val*-,* LimInf*- or* LimSup*-automaton whose runs yield monotonic weight sequences.*

Proof. Consider a Sup-automaton $\mathcal{A} = (\Sigma, Q, \iota, \delta)$. The idea is to construct an equivalent Sup-automaton $\mathcal{A}'$ that memorizes the maximal visited weight, and optionally take it as a LimInf- or LimSup-automaton. A similar construction appears in [CDH10b, Lem. 1] where for every run of $\mathcal{A}$ there is a run of $\mathcal{A}'$ yielding a weight sequence that is eventually constant, but it is not necessarily the case that every run of $\mathcal{A}'$ has a monotonic weight sequence. Let V be the set of weights on $\mathcal{A}$'s transitions. Since $|V| < \infty$, we can fix the minimal weight $v_0 = \min(V)$. We construct $\mathcal{A}' = (\Sigma, Q \times V, (\iota, v_0), \delta')$ where $\delta' \colon (Q \times V) \times \Sigma \to 2^{Q \times V}$ is defined as follows. Given $p \in Q$, $v, v' \in V$, and $\sigma \in \Sigma$, we have that $(v', (q, \max\{v, v'\})) \in \delta'((p, v), \sigma)$ if and only if $(v', q) \in \delta(p, \sigma)$. Notice that if $\mathcal{A}$ is deterministic, so is $\mathcal{A}'$. Clearly, the Sup-automata $\mathcal{A}'$ and $\mathcal{A}$ are equivalent, and the construction of $\mathcal{A}'$ is in PTIME in the size of $\mathcal{A}$. Observe that, by construction, every run ρ of $\mathcal{A}'$ yields a nondecreasing weight sequence for which there exists $i \in \mathbb{N}$ such that for all $j \geq i$ we have $\gamma(\rho[i]) = \gamma(\rho[j]) = \mathsf{Sup}(\gamma(\rho))$. Hence, $\mathcal{A}'$ can be equivalently interpreted as a Sup-, LimInf or LimSup-automaton. The construction for a given Inf-automaton is dual as it consists in memorizing the minimal visited weight, therefore the weight sequences are nonincreasing. □

We show that the common classes of quantitative automata always express $\sup$-closed properties, which will simplify the study of their safety and liveness.

Proposition 5.7.2. *Let* Val $\in \{\mathsf{Inf}, \mathsf{Sup}, \mathsf{LimInf}, \mathsf{LimSup}, \mathsf{LimInfAvg}, \mathsf{LimSupAvg}, \mathsf{DSum}\}$. *Every* Val*-automaton expresses a property that is* $\sup$*-closed. Furthermore its top value is rational, attainable by a run, and can be computed in* PTIME.

Proof. Observe that, by Theorem 5.7.1 the cases of Val $\in \{\mathsf{Inf}, \mathsf{Sup}\}$ reduce to Val $\in \{\mathsf{LimInf}, \mathsf{LimSup}\}$. So, we can assume that Val $\in \{$LimInf, LimSup, LimInfAvg, LimSupAvg, DSum$\}$.

It is shown in the proof of [CDH10b, Thm. 3] that the top value of every Val-automaton $\mathcal{A}$ is attainable by a lasso run, and is therefore rational, and can be computed in PTIME. It is left to show that $\mathcal{A}$ is $\sup$-closed, meaning that for every finite word $u \in \Sigma^*$, there exists $\hat{w} \in \Sigma^\omega$, such that $\mathcal{A}(u\hat{w}) = \sup_{w'} \mathcal{A}(uw')$.

Let U be the set of states that $\mathcal{A}$ can reach running on u. Observe that for every state $q \in U$, we have that $\mathcal{A}^q$ is also a Val-automaton. Thus, by the above result, its top value $\top_q$ is attainable by a run on some word w_q. Hence, for Val $\in \{\mathsf{LimInf}, \mathsf{LimSup}, \mathsf{LimInfAvg}, \mathsf{LimSupAvg}\}$, we have $\hat{w} = w_q$, such that $\top_q = \max(\top_{q'} \mid q' \in U)$. For Val $\in \{\mathsf{DSum}\}$ with a discount factor λ, let P_q be the maximal accumulated value of a run of $\mathcal{A}$ on u that ends in the state q. Then, we have $\hat{w} = w_q$, such that $P_q + \lambda^{|u|} \cdot \top_q = \max(P_{q'} + \lambda^{|u|} \cdot \top_{q'} \mid q' \in U)$. □

5.8 Subroutine: Constant-Function Check

We will show that the problems of whether a given automaton is safe or live are closely related to the problem of whether an automaton expresses a constant function, motivating its study

in this section. We first prove the problem hardness by reduction from the universality of nondeterministic finite-state automata (NFAs) and reachability automata.

Lemma 5.8.1. *Let* Val $\in \{$Sup, Inf, LimInf, LimSup, LimInfAvg, LimSupAvg, DSum$\}$. *Deciding whether a* Val*-automaton $\mathcal{A}$ expresses a constant function is* PSpace*-hard.*

Proof. First, we prove the case where Val $\in \{$Inf, LimInf, LimSup, LimInfAvg, LimSupAvg, DSum$\}$. The proof goes by reduction from the universality problem of nondeterministic finite-state automata (NFAs), which is known to be PSpace-complete. Consider an NFA $\mathcal{A} = (\Sigma, Q, \iota, F, \delta)$ over the alphabet $\Sigma = \{a, b\}$. We construct in PTime a Val-automaton $\mathcal{A}' = (\Sigma_\#, Q', \iota, \delta')$ over the alphabet $\Sigma_\# = \{a, b, \#\}$, such that $\mathcal{A}$ is universal if and only if $\mathcal{A}'$ is constant. $\mathcal{A}'$ has two additional states, $Q' = Q \uplus \{q_0, q_1\}$, and its transition function δ' is defined as follows:

- For every $(q, \sigma, p) \in \delta$, we have $q \xrightarrow{\sigma:1} p$.
- For every $q \in Q \setminus F$, we have $q \xrightarrow{\#:0} q_0$.
- For every $q \in F$, we have $q \xrightarrow{\#:1} q_1$.
- For every $\sigma \in \Sigma \cup \{\#\}$, we have $q_0 \xrightarrow{\sigma:0} q_0$, and $q_1 \xrightarrow{\sigma:1} q_1$.

Let $\top$ be the top value of $\mathcal{A}'$. (We have $\top = 1$ in all cases, except for Val = DSum.) First, note that for every word w with no occurrence of $\#$, we have that $\mathcal{A}'(w) = \top$, as all runs of $\mathcal{A}'$ visit only transitions with weight 1. If $\mathcal{A}$ is not universal, then there exists a word $u \in \{a, b\}^*$ such that $\mathcal{A}$ has no run over u from ι to some accepting state, and thus all runs of $\mathcal{A}'$ over $u\#$ from ι reach q_0. Hence, $\mathcal{A}'(u\#a^\omega) \neq \top$, while $\mathcal{A}'(a^\omega) = \top$, therefore $\mathcal{A}'$ is not constant. Otherwise, namely when $\mathcal{A}$ is universal, all infinite words with at least one occurrence of $\#$ can reach q_1 while only visiting 1-weighted transitions, and thus $\mathcal{A}'(w) = \top$ for all $w \in \{a, b, \#\}^\omega$.

Next, we prove the case where Val = Sup. The proof goes by reduction from the universality problem of a complete reachability automaton $\mathcal{A}'$ (i.e., a complete Büchi automaton all of whose states are rejecting, except for a single accepting sink). The problem is known to be PSpace-hard by a small adaptation to the standard reduction from the problem of whether a given Turing machine T that uses a polynomial working space accepts a given word u to NFA universality[1]. By this reduction, if T accepts u then $\mathcal{A}'$ accepts all infinite words, and if T does not accept u then $\mathcal{A}'$ accepts some words, while rejecting others by arriving in all runs to a rejecting sink after a bounded number of transitions. As a complete reachability automaton $\mathcal{A}'$ can be viewed as special case of a Sup-automaton $\mathcal{A}$, where transitions to nonaccepting states have weight 0 and to accepting states have weight 1, the hardness result directly follows to whether a Sup-automaton is constant. □

A simple solution to the problem is to check whether the given automaton $\mathcal{A}$ is equivalent to an automaton $\mathcal{B}$ expressing the constant top value of $\mathcal{A}$, which is computable in PTime by Theorem 5.7.2. For some automata classes, it suffices for a matching upper bound.

Proposition 5.8.2. *Deciding whether an* Inf*-,* Sup*-,* LimInf*-, or* LimSup*-automaton expresses a constant function is* PSpace*-complete.*

[1] Due to private communication with Christof Löding. See also [KZ17, Thm. A.1].

Proof. PSPACE-hardness is shown in Theorem 5.8.1. For the upper bound, we compute in PTIME, due to Theorem 5.7.2, the top value $\top$ of the given automaton $\mathcal{A}$, construct in constant time an automaton $\mathcal{B}$ of the same type as $\mathcal{A}$ that expresses the constant function $\top$, and check whether $\mathcal{A}$ and $\mathcal{B}$ are equivalent. This equivalence check is in PSPACE for arbitrary automata of the considered types [CDH10b, Thm. 4]. □

Yet, this simple approach does not work for DSum-automata, whose equivalence is an open problem, and for limit-average automata, whose equivalence is undecidable [DDG+10, CDE+10, HPPR18].

For DSum-automata, our alternative solution removes "nonoptimal" transitions from the automaton and then reduces the problem to the universality problem of NFAs.

Theorem 5.8.3. *Deciding whether a* DSum*-automaton expresses a constant function is* PSPACE*-complete.*

Proof. PSPACE-hardness is shown in Theorem 5.8.1. Consider a DSum-automaton $\mathcal{A}$. By Theorem 5.7.2, for every state q of $\mathcal{A}$ we can compute in PTIME the top value $\top_q$ of $\mathcal{A}^q$. We then construct in PTIME a DSum-automaton $\mathcal{A}'$, by removing from $\mathcal{A}$ every transition $q \xrightarrow{\sigma:x} q'$, for which $x + \lambda \cdot \top_{q'} < \top_q$. Finally, we consider $\mathcal{A}'$ as an (incomplete) NFA $\mathcal{A}''$ all of whose states are accepting.

We claim that $\mathcal{A}''$ is universal, which is checkable in PSPACE, if and only if $\mathcal{A}$ expresses a constant function. Indeed, if $\mathcal{A}''$ is universal then for every word w, there is a run of $\mathcal{A}''$ on every prefix of w. Thus, by König's lemma there is also an infinite run on w along the transitions of $\mathcal{A}''$. Therefore, there is a run of $\mathcal{A}$ on w that forever follows optimal transitions, namely ones that guarantee a continuation with the top value. Hence, by the discounting of the value function, the value of this run converges to the top value. If $\mathcal{A}''$ is not universal, then there is a finite word u for which all runs of $\mathcal{A}$ on it reach a dead-end state. Thus, all runs of $\mathcal{A}$ on u must have a transition $q \xrightarrow{\sigma:x} q'$, for which $x + \lambda \cdot \top_{q'} < \top_q$, implying that no run of $\mathcal{A}$ on a word w for which u is a prefix can have the top value. □

The solution for limit-average automata is more involved. It is based on a reduction to the limitedness problem of distance automata, which is known to be in PSPACE [Has82, Sim94, Has00, LP04]. We start by presenting Johnson's algorithm, which we will use for manipulating the transition weights of the given automaton, and proving some properties of distance automata, which we will need for the reduction.

A *weighted graph* is a directed graph $G = \langle V, E \rangle$ equipped with a weight function $\gamma : E \to \mathbb{Z}$. The cost of a path $p = v_0, v_1, \ldots, v_k$ is $\gamma(p) = \sum_{i=0}^{k-1} \gamma(v_i, v_{i+1})$.

Proposition 5.8.4 (Johnson's Algorithm [Joh77, Lem. 2 and Thms. 4 and 5])**.** *Consider a weighted graph $G = \langle V, E \rangle$ with weight function $\gamma : E \to \mathbb{Z}$, such that G has no negative cycles according to γ. We can compute in* PTIME *functions $h : V \to \mathbb{Z}$ and $\gamma' : E \to \mathbb{N}$ such that for every path $p = v_0, v_1, \ldots, v_k$ in G it holds that $\gamma'(p) = \gamma(p) + h(v_0) - h(v_k)$.*

Remark 5.8.5. *Theorem 5.8.4 is stated for graphs, while we will apply it for graphs underlying automata, which are multi-graphs, namely having several transitions between the same pairs of states. Nevertheless, to see that Johnson's algorithm holds also in our case, one can change every automaton to an equivalent one whose underlying graph is a standard graph, by splitting every state into several states, each having a single incoming transition.*

A *distance automaton* is a weighted automaton over the tropical semiring (a.k.a., min-plus semiring) with weights in $\{0, 1\}$. It can be viewed as a quantitative automaton over finite words with transition weights in $\{0, 1\}$ and the value function of summation, extended with accepting states. A distance automaton is of *limited distance* if there exists a bound on the automaton's values on all accepted words.

Lifting limitedness to infinite words, we have by König's lemma that a total distance automaton of limited distance b, in which all states are accepting, is also guaranteed to have a run whose weight summation is bounded by b on every infinite word.

Proposition 5.8.6. *Consider a total distance automaton $\mathcal{D}$ of limited distance b, in which all states are accepting. Then, for every infinite word w, there exists an infinite run of $\mathcal{D}$ on w whose summation of weights (considering only the transition weights and ignoring the final weights of states) is bounded by b.*

Proof. Consider an infinite word w, and let T be the tree of $\mathcal{D}$'s runs on prefixes of w whose values are bounded by b. Notice that T is an infinite tree since, by the totalness of $\mathcal{D}$ and the fact that all states are accepting, for every prefix of w there is at least one such run. As the branching degree of T is bounded by the number of states in $\mathcal{D}$, there exists by König's lemma an infinite branch ρ in T. Observe that the summation of weights along ρ is bounded by b—were it not the case, there would have been a position in ρ up to which the summation has exceeded b, contradicting the definition of T. □

Lifting nonlimitedness to infinite words, it may not suffice for our purposes to have an infinite word on which all runs of the distance automaton are unbounded, as their limit-average value might still be 0. Yet, thanks to the following lemma, we are able to construct an infinite word on which the limit-average value is strictly positive.

Lemma 5.8.7. *Consider a total distance automaton $\mathcal{D}$ of unlimited distance, in which all states are accepting. Then, there exists a finite nonempty word u such that $\mathcal{D}(u) = 1$ and the possible runs of $\mathcal{D}$ on u lead to a set of states U such that the distance automaton that is the same as $\mathcal{D}$ but with U as the set of its initial states is also of unlimited distance.*

Proof. Let Q be the set of states of $\mathcal{D}$. For a set $S \subseteq Q$, we denote by $\mathcal{D}^S$ the distance automaton that is the same as $\mathcal{D}$ but with S as the set of its initial states. Let B be the set of sets of states from which $\mathcal{D}$ is of limited distance. That is, $B = \{S \subseteq Q \mid \mathcal{D}^S \text{ is of limited distance}\}$. If $B = \emptyset$, the statement directly follows.

Otherwise, $B \neq \emptyset$. Since for all $S \in B$, the distance automaton $\mathcal{D}^S$ is bounded, we can define $\hat{b}$ as the minimal number, such that for every $S \in B$ and finite word u, we have $\mathcal{D}^S(u) \leq \hat{b}$. Formally, $\hat{b} = \max_{S \in B}(\min\{b \in \mathbb{N} \mid \forall u \in \Sigma^*, \mathcal{D}^S(u) \leq b\})$. Because $\mathcal{D}$ is of unlimited distance, we can exhibit a finite word mapped by $\mathcal{D}$ to an arbitrarily large value. In particular, there exists a word z such that $\mathcal{D}(z) \geq \hat{b} + 2$, i.e., the summation of the weights along every run of $\mathcal{D}$ on z is at least $\hat{b}+2$. Additionally, because transitions are weighted over $\{0, 1\}$, there exists at least one prefix $x \preceq z$ for which $\mathcal{D}(x) = 1$. Let the finite word y be such that $z = xy$. Next, we prove that x fulfills the statement, namely that the distance automaton $\mathcal{D}^X$, where X is the set of states that $\mathcal{D}$ can reach with runs on x, is also of unlimited distance. Assume towards contradiction that $X \in B$. By construction of B, we have that $\mathcal{D}^X$ is of limited distance. In fact, $\mathcal{D}^X(u) \leq \hat{b}$ for all finite words u, by the definition of $\hat{b}$. Hence $\mathcal{D}^X(y) \leq \hat{b}$, implying that $\mathcal{D}(z) = \mathcal{D}(xy) \leq \hat{b} + 1$, leading to a contradiction, as $\mathcal{D}(z) \geq \hat{b} + 2$. □

Using Theorems 5.8.4, 5.8.6 and 5.8.7 we are in position to solve our problem by reduction to the limitedness problem of distance automata.

Theorem 5.8.8. *Deciding whether a* LimInfAvg- *or* LimSupAvg-*automaton expresses a constant function, for a given constant or any constant, is* PSPACE*-complete.*

Proof. PSPACE-hardness is shown in Theorem 5.8.1. Consider a LimInfAvg- or LimSupAvg-automaton $\mathcal{A}$. We provide the upper bound as follows. First we construct in polynomial time a distance automaton $\mathcal{D}$, and then we reduce our statement to the limitedness problem of $\mathcal{D}$, which is decidable in PSPACE [Sim94].

By Theorem 5.7.2, one can first compute in polynomial time the top value of $\mathcal{A}$ denoted by $\top$. Thus, $\mathcal{A}$ expresses an arbitrary constant if and only if it expresses the constant function $\top$. From $\mathcal{A}$, we construct the automaton $\mathcal{A}'$, by subtracting $\top$ from all transitions weights (by Theorem 5.7.2, $\top$ is guaranteed to be rational). By construction the top value of $\mathcal{A}'$ is 0, i.e., $\mathcal{A}'(w) \leq 0$ for all w, and the question to answer is whether $\mathcal{A}'$ expresses the constant function 0, namely whether or not exists some word w such that $\mathcal{A}'(w) < 0$.

Next, we construct from $\mathcal{A}'$, in which the nondeterminism is resolved by $\sup$ as usual, the opposite automaton $\mathcal{A}''$, in which the nondeterminism is resolved by $\inf$, by changing every transition weight x to $-x$. If $\mathcal{A}'$ is a LimInfAvg-automaton then $\mathcal{A}''$ is a LimSupAvg-automaton, and vice versa. Observe that for every word w, we have $\mathcal{A}'(w) = -\mathcal{A}''(w)$. Now, we shall thus check if there exists a word w, such that $\mathcal{A}''(w) > 0$.

Since for every word w, we have that $\mathcal{A}''(w) \geq 0$, there cannot be a reachable cycle in $\mathcal{A}''$ whose average value is negative. Otherwise, some run would have achieved a negative value, and as the nondeterminism of $\mathcal{A}''$ is resolved with $\inf$, some word would have been mapped by $\mathcal{A}'$ to a negative value. Yet, there might be in $\mathcal{A}''$ transitions with negative weights. Thanks to Johnson's algorithm [Joh77] (see Theorem 5.8.4 and the remark after it), we can construct from $\mathcal{A}''$ in polynomial time an automaton $\mathcal{A}'''$ that resolves the nondeterminism as $\mathcal{A}''$ and is equivalent to it, but has no negative transition weights. It is worth emphasizing that since the value of the automaton on a word is defined by the limit of the average values of forever growing prefixes, the bounded initial and final values that result from Johnson's algorithm have no influence.

Finally, we construct from $\mathcal{A}'''$ the automaton $\mathcal{B}$ (of the same type), by changing every strictly positive transition weight to 1. So, $\mathcal{B}$ has transitions weighted over $\{0, 1\}$. Observe that while $\mathcal{A}'''$ and $\mathcal{B}$ need not be equivalent, for every word w, we have $\mathcal{A}'''(w) > 0$ if and only if $\mathcal{B}(w) > 0$. This is because $x \cdot \mathcal{B}(w) \leq \mathcal{A}'''(w) \leq y \cdot \mathcal{B}(w)$, where x and y are the minimal and maximal strictly positive transition weights of $\mathcal{A}'''$, respectively. Further, we claim that $\mathcal{B}$ expresses the constant function 0 if and only if the distance automaton $\mathcal{D}$, which is a copy of $\mathcal{B}$ where all states are accepting, is limited.

If $\mathcal{D}$ is limited, then by Theorem 5.8.6 there is a bound b, such that for every infinite word w, there exists an infinite run of $\mathcal{D}$ (and of $\mathcal{B}$) over w whose summation of weights is bounded by b. Thus, the value of $\mathcal{B}$ (i.e., LimInfAvg or LimSupAvg) for this run is 0.

If $\mathcal{D}$ is not limited, observe that the existence of an infinite word on which all runs of $\mathcal{D}$ are of unbounded value does not suffice to conclude. Indeed, the run that has weight 1 only in positions $\{2^n \mid n \in \mathbb{N}\}$ has a limit-average of 0. Nevertheless, we are able to provide a word w, such that the LimInfAvg and LimSupAvg values of every run of $\mathcal{B}$ over w are strictly positive.

By Theorem 5.8.7, there exists a finite nonempty word u_1, such that $\mathcal{D}(u_1) = 1$ and the possible runs of $\mathcal{D}$ over u lead to a set of states S_1, such that the distance automaton $\mathcal{D}^{S_1}$ (defined as $\mathcal{D}$ but where S_1 is the set of initial states) is of unlimited distance. We can then apply Theorem 5.8.7 on $\mathcal{D}^{S_1}$, getting a finite nonempty word u_2, such that $\mathcal{D}^{S_1}(u_2) = 1$, and the runs of $\mathcal{D}^{S_1}$ over u_2 lead to a set S_2, such that $\mathcal{D}^{S_2}$ is of unlimited distance, and so on. Since there are finitely many subsets of states of $\mathcal{D}$, we reach a set S_ℓ, such that there exists $j < \ell$ with $S_j = S_\ell$. We define the infinite word $w = u_1 \cdot u_2 \cdots u_j \cdot (u_{j+1} \cdot u_{j+2} \cdots u_\ell)^\omega$. Let m be the maximum length of u_i, for $i \in [1, \ell]$. Next, we show that the LimInfAvg and LimSupAvg values of every run of $\mathcal{D}$ (and thus the value of $\mathcal{B}$) over w is at least $\frac{1}{m}$.

Indeed, consider any infinite run ρ of $\mathcal{D}$ over w. At position $|u_1|$, the summation of weights of ρ is at least 1, so the average is at least $\frac{1}{m}$. Since the run ρ at this position is in some state $q \in S_1$ and $\mathcal{D}^{S_1}(u_2) = 1$, the continuation until position $|u_1 u_2|$ will go through at least another 1-valued weight, having the average at position $|u_1 u_2|$ is at least $\frac{1}{m}$. Then, for every position k and natural number $i \in \mathbb{N}$ such that $|u_1 \cdots u_i| \leq k < |u_1 \cdots u_{i+1}|$, we have $\frac{i-1}{i \cdot m} \leq \frac{i}{k} \leq \frac{i}{i \cdot m} = \frac{1}{m}$. Therefore, as i goes to infinity, the running average of weights of ρ converges to $\frac{1}{m}$. □

5.9 Safety of Quantitative Automata

For studying the safety of automata, we build on the alternative characterizations of quantitative safety through threshold safety and continuity, as discussed in Sections 5.3.1 and 5.3.2. The characterizations for totally-ordered value domains hold in particular for properties expressed by quantitative automata. First, we extend the notion of safety from properties to value functions, allowing us to characterize families of safe quantitative automata. Finally, we provide algorithms to construct the safety closure of a given automaton $\mathcal{A}$ and to decide whether $\mathcal{A}$ is safe.

5.9.1 Safety of Value Functions

In this section, we focus on the value functions of quantitative automata, which operate on the value domain of real numbers. In particular, we carry the definitions of safety, co-safety, and discounting to value functions. This allows us to characterize safe (resp. co-safe, discounting) value functions as those for which all automata with this value function are safe (resp. co-safe, discounting). Moreover, we characterize discounting value functions as those that are safe and co-safe.

Recall that we consider the value functions of quantitative automata to be bounded from below and above for every finite input domain $V \subset \mathbb{Q}$. As the set V^ω can be taken as a Cantor space, just like Σ^ω, we can carry the notions of safety, co-safety, and discounting from properties to value functions.

Definition 5.9.1 (Safety and co-safety of value functions)**.** *A value function* $\mathsf{Val} : \mathbb{Q}^\omega \to \mathbb{R}$ *is* safe *when for every finite subset* $V \subset \mathbb{Q}$*, infinite sequence* $x \in V^\omega$*, and value* $v \in \mathbb{R}$*, if* $\mathsf{Val}(x) < v$ *then there exists a finite prefix* $z \prec x$ *such that* $\sup_{y \in V^\omega} \mathsf{Val}(zy) < v$*. Similarly, a value function* $\mathsf{Val} : \mathbb{Q}^\omega \to \mathbb{R}$ *is* co-safe *when for every finite subset* $V \subset \mathbb{Q}$*, infinite sequence* $x \in V^\omega$*, and value* $v \in \mathbb{R}$*, if* $\mathsf{Val}(x) > v$ *then there exists a finite prefix* $z \prec x$ *such that* $\inf_{y \in V^\omega} \mathsf{Val}(zy) > v$*.*

Definition 5.9.2 (Discounting value function)**.** *A value function* Val : $\mathbb{Q}^\omega \to \mathbb{R}$ *is* discounting *when for every finite subset* $V \subset \mathbb{Q}$ *and every* $\varepsilon > 0$ *there exists* $n \in \mathbb{N}$ *such that for every* $x \in V^n$ *and* $y, y' \in V^\omega$ *we have* $|\mathsf{Val}(xy) - \mathsf{Val}(xy')| < \varepsilon$*.*

We remark that by Theorems 5.4.5 and 5.4.8, the value function Inf is safe and Sup is co-safe; moreover, DSum is discounting by definition. Now, we characterize the safety (resp. co-safety) of a given value function by the safety (resp. co-safety) of the automata family it defines. We emphasize that the proofs of the two statements are not dual. In particular, exhibiting a finite set of weights that falsifies the safety of a value function from a nonsafe automaton requires a compactness argument.

Theorem 5.9.3. *Consider a value function* Val*. All* Val*-automata are safe (resp. co-safe) iff* Val *is safe (resp. co-safe).*

Proof. We show the case of safety and co-safety separately as they are not symmetric due to nondeterminism of automata.

Co-safety. One direction is immediate, by constructing a deterministic automaton that expresses the value function itself: If Val is not co-safe then there exists some finite set $V \subset \mathbb{Q}$ of weights with respect to which it is not co-safe. Consider the deterministic Val-automaton over the alphabet V with a single state and a self loop with weight $v \in V$ over every letter $v \in V$, that is, the letters coincide with the weights. Then, the automaton simply expresses Val and is therefore not co-safe.

For the other direction, consider a co-safe value function Val, a Val-automaton $\mathcal{A}$ over an alphabet Σ with a set of weights $V \subset \mathbb{Q}$, a value $v \in \mathbb{R}$, and a word w, such that $\mathcal{A}(w) > v$. We need to show that there exists a prefix $u \prec w$ such that $\inf_{w' \in \Sigma^\omega} \mathcal{A}(uw') > v$. Let ρ be some run of $\mathcal{A}$ on w such that $\mathsf{Val}(\gamma(\rho)) > v$. (Observe that such a run exists, since the value domain is totally ordered, as the supremum of runs that are not strictly bigger than v is also not bigger than v.)

Then, by the co-safety of Val, there exists a prefix ρ' of ρ, such that $\inf_{x' \in V^\omega} \mathsf{Val}(\gamma(\rho')x') > v$. Let $u \prec w$ be the prefix of w of length $|\rho'|$. By the completeness of $\mathcal{A}$, for every word $w'' \in \Sigma^\omega$ there exists a run $\rho'\rho''$ over uw'', and by the above we have $\mathsf{Val}(\gamma(\rho'\rho'')) > v$. Since $\mathcal{A}(uw'') \geq \mathsf{Val}(\gamma(\rho'\rho''))$, it follows that $\inf_{w' \in \Sigma^\omega} \mathcal{A}(uw') \geq \inf_{x' \in V^\omega} \mathsf{Val}(\gamma(\rho')x') > v$, as required.

Safety. One direction is immediate: if the value function is not safe, we get a nonsafe automaton by constructing a deterministic automaton that expresses the value function itself, as detailed in the case of co-safety.

As for the other direction, consider a nonsafe Val-automaton $\mathcal{A}$ over an alphabet Σ with a finite set $V \subset \mathbb{Q}$ of weights. Then, there exist a value $v \in \mathbb{R}$ and a word w with $\mathcal{A}(w) < v$, such that for every prefix $u \prec w$, we have $\sup_{w' \in \Sigma^\omega} \mathcal{A}(uw') \geq v$. Let $v' \in (\mathcal{A}(w), v)$ be a value strictly between $\mathcal{A}(w)$ and v. For every prefix $u \prec w$ of length $i > 0$, let $w_i \in \Sigma^\omega$ be an infinite word and r_i a run of $\mathcal{A}$ on uw_i, such that the value of r_i is at least v'. Such a run exists since for all $u \prec w$, the supremum of runs on uw', where $w' \in \Sigma^\omega$, is larger than v'.

Let r' be a run of $\mathcal{A}$ on w, constructed in the spirit of König's lemma by inductively adding transitions that appear in infinitely many runs r_i. That is, the first transition t_0 on $w[0]$ in r' is chosen such that t_0 is the first transition of r_i for infinitely many $i \in \mathbb{N}$. Then t_1 on $w[1]$,

is chosen such that $t_0 \cdot t_1$ is the prefix of r_i for infinitely many $i \in \mathbb{N}$, and so on. Let ρ be the sequence of weights induced by r'. Observe that $\mathsf{Val}(\rho) \leq \mathcal{A}(w) < v'$. Now, every prefix $\eta \prec \rho$ of length i is also a prefix of the sequence ρ_i of weights induced by the run r_i, and by the above construction, we have $\mathsf{Val}(\rho_i) \geq v'$. Thus, while $\mathsf{Val}(\rho) < v'$, for every prefix $\eta \prec \rho$, we have $\sup_{\rho' \in V^\omega} \mathsf{Val}(\eta\rho') \geq v'$, implying that Val is not safe. □

Recall that a value function together with a finite set $V \subset \mathbb{Q}$ of weights can be seen as a quantitative property over the finite alphabet $\Sigma = V$. Then, thanks to Theorem 5.3.27, we can characterize discounting value functions as those that are both safe and co-safe.

Corollary 5.9.4. *A value function is discounting iff it is safe and co-safe.*

As a consequence of Theorems 5.9.3 and 5.9.4, we obtain the following.

Corollary 5.9.5. *All* Val*-automata are discounting iff* Val *is discounting.*

5.9.2 Deciding Safety of Quantitative Automata

We now switch our focus from generic value functions to families of quantitative automata defined by the common value functions Inf, Sup, LimInf, LimSup, LimInfAvg, LimSupAvg, and DSum. As remarked in Section 5.9.1, the value functions Inf and DSum are safe, thus all Inf-automata and DSum-automata express a safety property by Theorem 5.9.3. Below, we focus on the remaining value functions of interest.

Given a Val-automaton $\mathcal{A}$ where Val is one of the nonsafe value functions above, we describe (i) a construction of an automaton that expresses the safety closure of $\mathcal{A}$, and (ii) an algorithm to decide whether $\mathcal{A}$ is safe. For these value functions, we can construct the safety closure as an Inf-automaton.

Theorem 5.9.6. *Let* $\mathsf{Val} \in \{\mathsf{Sup}, \mathsf{LimInf}, \mathsf{LimSup}, \mathsf{LimInfAvg}, \mathsf{LimSupAvg}\}$. *Given a* Val*-automaton $\mathcal{A}$, we can construct in* PTIME *an* Inf*-automaton $\mathcal{A}'$ that expresses its safety closure. Moreover, if $\mathcal{A}$ is deterministic, then so is $\mathcal{A}'$.*

Proof. Let $\mathcal{A} = (\Sigma, Q, \iota, \delta)$ be a Val-automaton as above, where $\mathsf{Val} \neq \mathsf{Sup}$. We construct an Inf-automaton $\mathcal{A}' = (\Sigma, Q, \iota, \delta')$ that expresses the safety closure of $\mathcal{A}$ by only changing the weights of $\mathcal{A}$'s transitions, as follows. For every state $q \in Q$, we compute in PTIME, due to Theorem 5.7.2, the top value $\top_q$ of the automaton $\mathcal{A}^q$. For every state $p \in Q$ and letter $\sigma \in \Sigma$, we define the transition function $\delta'(p, \sigma) = \{(\top_q, q) \mid \exists x \in \mathbb{Q} : (x, q) \in \delta(p, \sigma)\}$. Notice that $\mathcal{A}$ and $\mathcal{A}'$ are identical except for their transition weights, therefore $\mathcal{A}'$ is deterministic if $\mathcal{A}$ is.

Consider a run ρ of $\mathcal{A}'$. Let $i \in \mathbb{N}$ be the number of transitions before ρ reaches its ultimate strongly connected component, i.e., the one ρ stays indefinitely. By construction of $\mathcal{A}'$, the sequence $\gamma(\rho)$ of weights is nonincreasing, and for all $j \geq i$ we have that $\gamma(\rho[i]) = \gamma(\rho[j])$. Again, by construction, the value $\gamma(\rho[j])$ is the maximal value $\mathcal{A}$ can achieve after the first j steps of ρ. Moreover, since $\gamma(\rho)$ is nonincreasing, it is the minimal value among the prefixes of $\gamma(\rho)$. In other words, $\gamma(\rho[i]) = \inf_{j \in \mathbb{N}} \sup_{\rho' \in R_j} \mathsf{Val}(\gamma(\rho[..j]\rho'))$ where R_j is the set of runs of $\mathcal{A}$ starting from the state reached after the finite run $\rho[..j]$. Notice that this defines exactly the value of the safety closure for the run ρ. Therefore, it is easy to see that $\mathcal{A}'(w) = \inf_{u \prec w} \sup_{w' \in \Sigma^\omega} \mathcal{A}(uw')$ for all $w \in \Sigma^\omega$.

For $\mathsf{Val} = \mathsf{Sup}$, we use Theorem 5.7.1 to first translate $\mathcal{A}$ to a LimInf- or LimSup-automaton, which preserves determinism as needed. □

Figure 5.2: A Sup-automaton $\mathcal{A}$ together with its safety closure $\mathcal{B}$ given as an Inf-automaton, which cannot be expressed by a Sup-automaton.

For the prefix-independent value functions we study, the safety-closure automaton from the proof of Theorem 5.9.6 can be taken as a deterministic automaton with the same value function.

Theorem 5.9.7. *Let* Val $\in \{$LimInf, LimSup, LimInfAvg, LimSupAvg$\}$. *Given a* Val*-automaton* $\mathcal{A}$*, we can construct in* PTime *a* Val*-automaton that expresses its safety closure and can be determinized in* ExpTime.

Proof. Let $\mathcal{A}$ be a Val-automaton. We construct its safety closure $\mathcal{A}'$ as an Inf-automaton in polynomial time, as in the proof of Theorem 5.9.6. Observe that, by construction, every run ρ of $\mathcal{A}'$ yields a nonincreasing weight sequence for which there exists $i \in \mathbb{N}$ such that for all $j \geq i$ we have $\gamma(\rho[i]) = \gamma(\rho[j]) = \mathsf{Inf}(\gamma(\rho))$. Then, to construct a Val-automaton that is equivalent to $\mathcal{A}'$, we simply copy $\mathcal{A}'$ and use the value function Val instead. Similarly, to obtain a deterministic Val-automaton that is equivalent to $\mathcal{A}'$, we first determinize the Inf-automaton $\mathcal{A}'$ in exponential time [KL07, Thm. 7], and then the result can be equivalently considered as a Val-automaton for the same reason as before. □

By contrast, this is not possible in general for Sup-automata, as Figure 5.2 witnesses.

Proposition 5.9.8. *Some* Sup*-automaton admits no* Sup*-automata that expresses its safety closure.*

Proof. Consider the Sup-automaton $\mathcal{A}$ given in Figure 5.2. We have $\mathit{SafetyCl}(\mathcal{A})(w) = 2$ if $w = a^\omega$ or the first c in w occurs before the first b in w (which may never occur), and $\mathit{SafetyCl}(\mathcal{A})(w) = 1$ otherwise. Suppose towards contradiction that there is a Sup-automaton $\mathcal{A}'$ expressing $\mathit{SafetyCl}(\mathcal{A})$. Since $\mathcal{A}'$ has finitely many weights, it is $\sup$-closed, and $\mathcal{A}'(a^\omega) = 2$, there is a run ρ of $\mathcal{A}'$ over a^ω in which the weight 2 occurs at least once, say at position i. Then, every valid continuation of the finite run $\rho[..i]$ over $\mathcal{A}'$ is mapped to at least 2. In particular, $\mathcal{A}'(a^i b^\omega) \geq 2$; however, $\mathit{SafetyCl}(\mathcal{A})(a^i b^\omega) = 1$. □

We first prove the hardness of deciding safety by reduction from constant-function checks.

Lemma 5.9.9. *Let* Val $\in \{$Sup, LimInf, LimSup, LimInfAvg, LimSupAvg$\}$. *It is* PSpace*-hard to decide whether a* Val*-automaton is safe.*

Proof. We can reduce in PTime the problem of whether a Val-automaton $\mathcal{A}$ with the top value $\top$ expresses a constant function, which is PSpace-hard by Theorem 5.8.1, to the problem of whether a Val-automaton $\mathcal{A}'$ is safe, by adding $\top$-weighted transitions over a fresh alphabet letter from all states of $\mathcal{A}$ to a new state $q_\top$, which has a $\top$-weighted self-loop over all alphabet letters.

Indeed, if $\mathcal{A}$ expresses the constant function $\top$, so does $\mathcal{A}'$ and it is therefore safe. Otherwise, $\mathcal{A}'$ is not safe, as a word w over $\mathcal{A}$'s alphabet for which $\mathcal{A}(w) \neq \top$ also has a value smaller

than $\top$ by $\mathcal{A}'$, while every prefix of it can be concatenated with a word that starts with the fresh letter, having the value $\top$. □

For automata classes with PSPACE equivalence check, a matching upper bound is straightforward by comparing the given automaton and its safety-closure automaton.

Theorem 5.9.10. *Deciding whether a* Sup-, LimInf-, *or* LimSup-*automaton expresses a safety property is* PSPACE*-complete.*

Proof. PSPACE-hardness is shown in Theorem 5.9.9. For the upper bound, we construct in PTIME, due to Theorem 5.9.7, the safety-closure automaton $\mathcal{A}'$ of the given automaton $\mathcal{A}$, and then check in PSPACE if $\mathcal{A} = \mathcal{A}'$. Notice that equivalence-check is in PSPACE for these value functions in general [CDH10b, Thm. 4]. □

On the other hand, even though equivalence of limit-average automata is undecidable [DDG+10, CDE+10, HPPR18]., we are able to provide a decision procedure using as a subroutine our algorithm to check whether a given limit-average automaton expresses a constant function (see Theorem 5.8.8). The key idea is to construct a limit-average automaton that expresses the constant function 0 iff the original automaton is safe. Our approach involves the determinization of the safety-closure automaton, resulting in an EXPSPACE complexity. Let us start with a lemma on checking the equivalence of limit-average automata.

Lemma 5.9.11. *Let* Val $\in \{\mathsf{LimInfAvg}, \mathsf{LimSupAvg}\}$ *and consider two* Val*-automata* $\mathcal{A}$ *and* $\mathcal{B}$*. If* $\mathcal{B}$ *is deterministic and each of its runs yields an eventually-constant weight sequence, deciding whether* $\mathcal{A}$ *and* $\mathcal{B}$ *are equivalent is in* PSPACE*.*

Proof. We construct $\mathcal{C}$ by taking the product between $\mathcal{A}$ and $\mathcal{B}$ where the weight of a transition in $\mathcal{C}$ is obtained by subtracting the weight of the corresponding transition in $\mathcal{B}$ from that in $\mathcal{A}$. We claim that $\mathcal{A}$ and $\mathcal{B}$ are equivalent iff $\mathcal{C}$ expresses the constant function 0. Indeed, consider a word $w \in \Sigma^\omega$. By definition, $\mathcal{A}(w) = \mathcal{B}(w)$ iff $\sup_{\rho_\mathcal{A} \in R_w^\mathcal{A}} \{\mathsf{Val}(\gamma(\rho_\mathcal{A}))\} - \mathsf{Val}(\gamma(\rho_\mathcal{B})) = 0$ where $\rho_\mathcal{B}$ is the unique run of $\mathcal{B}$ on w. Equivalently $\sup_{\rho_\mathcal{A} \in R_w^\mathcal{A}} \{\mathsf{Val}(\gamma(\rho_\mathcal{A})) - \mathsf{Val}(\gamma(\rho_\mathcal{B}))\} = 0$. We claim that $\mathsf{Val}(\gamma(\rho_\mathcal{A})) - \mathsf{Val}(\gamma(\rho_\mathcal{B})) = \mathsf{Val}(\gamma(\rho_\mathcal{A}) - \gamma(\rho_\mathcal{B}))$ where $\gamma(\rho_\mathcal{A}) - \gamma(\rho_\mathcal{B})$ is the sequence obtained by taking the elementwise difference of the weight sequences produced by the runs $\rho_\mathcal{A}$ and $\rho_\mathcal{B}$. This claim does not hold for arbitrary sequences of weights, but it does hold if the sequence of weights $\gamma(\rho_B)$ is eventually constant and Val is prefix independent. As the weight sequence of $\rho_\mathcal{B}$ is eventually constant by our initial assumption and Val is prefix independent, we can subtract elementwise from the weight sequence of each run of $\mathcal{A}$ that of $\mathcal{B}$. Thus, we get $\sup_{\rho_\mathcal{A} \in R_w^\mathcal{A}} \{\mathsf{Val}(\gamma(\rho_\mathcal{A})) - \mathsf{Val}(\gamma(\rho_\mathcal{B}))\} = 0$ iff $\sup_{\rho_\mathcal{A} \in R_w^\mathcal{A}} \{\mathsf{Val}(\gamma(\rho_\mathcal{A}) - \gamma(\rho_\mathcal{B}))\} = 0$. Observe that, by construction, each run of $\mathcal{C}$ produces a weight sequence that corresponds to this difference. Then, $\sup_{\rho_\mathcal{A} \in R_w^\mathcal{A}} \{\mathsf{Val}(\gamma(\rho_\mathcal{A}) - \gamma(\rho_\mathcal{B}))\} = 0$ iff $\sup_{\rho_\mathcal{C} \in R_w^\mathcal{C}} \{\mathsf{Val}(\gamma(\rho_\mathcal{C}))\} = 0$ iff $\mathcal{C}(w) = 0$. Finally, to check the equivalence of $\mathcal{A}$ and $\mathcal{B}$, we can decide by Theorems 5.7.2 and 5.8.8 if $\mathcal{C}(w) = 0$ for all $w \in \Sigma^\omega$. □

Using the lemma above, we obtain an algorithm to check whether a given limit-average automaton is safe.

Theorem 5.9.12. *Deciding whether a* LimInfAvg- *or* LimSupAvg*-automaton expresses a safety property is in* EXPSPACE*.*

Proof. Let Val $\in \{\mathsf{LimInfAvg}, \mathsf{LimSupAvg}\}$ and let $\mathcal{A}$ be a Val-automaton. We construct the safety-closure automaton of $\mathcal{A}$ whose weight sequences are eventually constant as in Theorem 5.9.6 and transform it into a deterministic Val-automaton $\mathcal{B}$ as in the proof of Theorem 5.9.7. To check the safety of $\mathcal{A}$, we can decide by Theorem 5.9.11 whether $\mathcal{A}$ and $\mathcal{B}$ are equivalent in PSPACE since $\mathcal{B}$ is deterministic and its weight sequences are eventually constant by construction. Because the construction of $\mathcal{B}$ might cause up to an exponential size blow-up, the decision procedure for checking the safety of limit-average automata is in $\mathrm{EXPSPACE}$. □

5.10 Liveness of Quantitative Automata

In this section, we provide algorithms to check liveness of quantitative automata, and to decompose them into a safety automaton and a liveness automaton. We build on the alternative characterizations of quantitative liveness, as discussed in Section 5.6.2. In particular, our algorithms take advantage of the fact that liveness and top liveness coincide for $\sup$-closed properties (Theorem 5.6.21).

5.10.1 Deciding Liveness of Quantitative Automata

Let us start with the problem of checking whether a quantitative automaton is live. We first provide a hardness result by reduction from constant-function checks.

Lemma 5.10.1. *Let* Val $\in \{\mathsf{Inf}, \mathsf{Sup}, \mathsf{LimInf}, \mathsf{LimSup}, \mathsf{LimInfAvg}, \mathsf{LimSupAvg}, \mathsf{DSum}\}$. *Deciding whether a* Val*-automaton $\mathcal{A}$ is live is* PSPACE*-hard.*

Proof. Let Val $\in \{\mathsf{Inf}, \mathsf{LimInf}, \mathsf{LimSup}, \mathsf{LimInfAvg}, \mathsf{LimSupAvg}, \mathsf{DSum}\}$ be a value function. Consider a Val-automaton $\mathcal{A}'$ that is constructed along the proofs of Theorem 5.8.1, in which we show that the constant-function check is PSPACE-hard. Observe that $\mathcal{A}'$ either (i) expresses the constant function $\top$, and is therefore live; or (ii) has a value $\top$ on some word w and a value $x < \top$ on some word w', where there is a prefix u of w', such that for every infinite word $\hat{w}$, we have $\mathcal{A}'(u\hat{w}) = x$, implying that $\mathcal{A}'$ is not live. Therefore, the PSPACE-hardness of the constant-function check extends to liveness-check.

The proof for Val = Sup goes by reduction from the constant-function check for Inf-automata, which is PSPACE-hard by Theorem 5.8.1. Given an Inf-automaton $\mathcal{A}$ over an alphabet $\Sigma = \{a, b\}$, we construct in PTIME a Sup-automaton $\mathcal{A}'$ such that $\mathcal{A}$ is constant iff $\mathcal{A}'$ is live.

First, using Theorem 5.7.1, we transform $\mathcal{A}$ into an equivalent Inf-automaton $\mathcal{B} = (\Sigma, Q_{\mathcal{B}}, \iota, \delta_{\mathcal{B}})$ whose runs are nonincreasing. Let $\mathcal{S}_{\mathcal{B}} = \{S_1, \ldots, S_k\}$ be the set of strongly connected components of $\mathcal{B}$. Note that, by construction, each $S \in \mathcal{S}_{\mathcal{B}}$ (for which there is a transition whose target is in S) is associated with a weight x such that all transitions whose target is in S has weight x, which we denote by $\gamma_{\mathcal{B}}(S) = x$ with a slight abuse of notation. Notice that $\gamma_{\mathcal{B}}(S)$ is undefined when S has no incoming transitions, which may happen if S is a trivial strongly connected component containing the initial or an unreachable state.

We now construct from $\mathcal{B}$ an Inf-automaton $\mathcal{C}$. The automaton $\mathcal{C}$ is a copy of $\mathcal{B}$ over the alphabet $\Sigma_{\#} = \Sigma \cup \{\#\}$ with two additional states $Q_{\mathcal{C}} = Q_{\mathcal{B}} \uplus \{q_0, q_1\}$, modified transition weights, and some additional transitions. The transition function $\delta_{\mathcal{C}}$ is defined as follows:

- For every transition $(q, \sigma, x, p) \in \delta_{\mathcal{B}}$ with $x \geq \top_{\mathcal{B}}$, we have $(q, \sigma, 1, p) \in \delta_{\mathcal{C}}$.

- For every transition $(q, \sigma, x, p) \in \delta_{\mathcal{B}}$ with $x < \top_{\mathcal{B}}$, we have $(q, \sigma, 0, p) \in \delta_{\mathcal{C}}$.
- For every $\sigma \in \Sigma \cup \{\#\}$, we have $(q_1, \sigma, 1, q_1) \in \delta_{\mathcal{C}}$ and $(q_0, \sigma, 0, q_0) \in \delta_{\mathcal{C}}$.
- For every strongly connected component $S \in \mathcal{S}_{\mathcal{B}}$ with $\gamma_{\mathcal{B}}(S) \geq \top_{\mathcal{B}}$ and for every $q \in S$, we have $(q, \#, 1, q_1) \in \delta_{\mathcal{C}}$.
- For every strongly connected component $S \in \mathcal{S}_{\mathcal{B}}$ with $\gamma_{\mathcal{B}}(S) < \top_{\mathcal{B}}$ and for every $q \in S$, we have $(q, \#, 0, q_0) \in \delta_{\mathcal{C}}$.

Notice that (i) we do not add transitions to q_0 or q_1 from strongly connected components for which the $\gamma_{\mathcal{B}}$ value is undefined, and (ii) by construction, the strongly connected components of $\mathcal{C}$ are given by the set $\mathcal{S}_{\mathcal{C}} = \{S_1, \ldots, S_k, T_0, T_1\}$ where, for $j \in \{0, 1\}$, we have $T_j = \{q_j\}$. Moreover, for every $S \in \mathcal{S}_{\mathcal{C}}$, we have $\gamma_{\mathcal{C}}(S) = 1$ if $S = T_1$ or $S \in \mathcal{S}_{\mathcal{B}}$ with $\gamma_{\mathcal{B}}(T) \geq \top_{\mathcal{B}}$, and $\gamma_{\mathcal{C}}(S) = 0$ otherwise.

We claim that $\mathcal{A}$ is constant iff $\mathcal{C}$ is constant. Since $\mathcal{A}$ and $\mathcal{B}$ are equivalent, we show that $\mathcal{B}$ is constant iff $\mathcal{C}$ is constant.

Assume $\mathcal{B}$ is constant, i.e., $\mathcal{B}(w) = \top_{\mathcal{B}}$ for all $w \in \Sigma^\omega$. Let w be a word with no occurrence of $\#$. There is a run of $\mathcal{B}$ over w such that every strongly connected component $S \in \mathcal{S}_{\mathcal{B}}$ it visits satisfies $\gamma_{\mathcal{B}}(S) \geq \top_{\mathcal{B}}$. By construction, $\mathcal{C}$ has a run over w following the same sequence of states, and thus the same strongly connected components, which satisfy $\gamma_{\mathcal{B}}(S) = 1$. Therefore, $\mathcal{C}(w) = 1$. Now, let w be a word with an occurrence of $\#$, i.e., $w = u\#w'$ for some $u \in \Sigma^*$ and $w' \in \Sigma^\omega_\#$. Since $\mathcal{B}$ is constant and an Inf-automaton, there is a finite run of $\mathcal{B}$ over u that always stays in strongly connected components that are weighted at least $\top_{\mathcal{B}}$. Then, $\mathcal{C}$ has a finite run over u staying only in 1-weighted components, reaching the 1-weighted bottom component T_1 after reading $u\#$, and thus $\mathcal{C}(w) = 1$. Therefore, $\mathcal{C}$ is also constant.

Assume $\mathcal{B}$ is not constant. Then, there exists $w_1, w_2 \in \Sigma^\omega$ such that $\mathcal{B}(w_1) < \mathcal{B}(w_2) = \top_{\mathcal{B}}$. By similar arguments as above, we have that $\mathcal{C}(w_2) = 1$. Moreover, all runs of $\mathcal{B}$ over w_1 ultimately stay in a strongly connected component for which the $\gamma_{\mathcal{B}}$ value is strictly less than $\top_{\mathcal{B}}$. Again, similarly as above, each of these runs correspond to a run of $\mathcal{C}$ over w_1, and each corresponding run ultimately stays in a strongly connected component for which the $\gamma_{\mathcal{C}}$ value is 0, and thus $\mathcal{C}(w_1) = 0$. Therefore, $\mathcal{C}$ is also not constant.

Now, we construct from the Inf-automaton $\mathcal{C}$ a Sup-automaton $\mathcal{A}'$. The automaton $\mathcal{A}'$ is a copy of $\mathcal{C}$ with the only difference being the transition weights: for every $(q, \sigma, x, p) \in \delta_{\mathcal{C}}$, we have $(q, \sigma, x', p) \in \delta_{\mathcal{A}'}$ where x' is the minimum over the values $\gamma_{\mathcal{C}}(S)$ such that the strongly connected component S is reachable from the state p. In other words, the weight of a transition in $\mathcal{A}'$ is 0 if some run starting from the target state can achieve the value 0 in $\mathcal{C}$, and it is 1 otherwise.

We claim that $\mathcal{C}$ expresses $SafetyCl(\mathcal{A}')$, which means $\mathcal{C}$ is constant iff $\mathcal{A}'$ is live, thanks to Theorems 5.6.21 and 5.7.2. First, observe that (i) $\mathcal{S}_{\mathcal{C}} = \mathcal{S}_{\mathcal{A}'}$, (ii) for every $S, S' \in \mathcal{S}_{\mathcal{C}}$, if S' is reachable from S and $\gamma_{\mathcal{C}}(S) = 0$, then $\gamma_{\mathcal{C}}(S') = \gamma_{\mathcal{A}'}(S') = 0$, and (iii) for every $S, S' \in \mathcal{S}_{\mathcal{C}}$, if S' is reachable from S and $\gamma_{\mathcal{C}}(S') = 1$, then $\gamma_{\mathcal{C}}(S) = 1$.

Consider a word $w \in \Sigma^\omega_\#$ such that $\mathcal{C}(w) = 0$. We want to show that $SafetyCl(\mathcal{A}')(w) = 0$, i.e., there is a prefix $u \prec w$ such that $\mathcal{A}'(uw') = 0$ for all $w' \in \Sigma^\omega_\#$. Since $\mathcal{C}(w) = 0$, every run of $\mathcal{C}$ over w ultimately stays in a strongly connected component S such that $\gamma_{\mathcal{C}}(S) = 0$. As $\mathcal{A}'$ only differs from $\mathcal{C}$ in transition weights, every run of $\mathcal{A}$ over w follows the same states and the strongly connected components. Notice that whenever such a run visits a strongly

connected component T with $\gamma_{\mathcal{C}}(T) = 1$, we have $\gamma_{\mathcal{A}'}(T) = 0$ by construction (as the same run ultimately reaches a component S with $\gamma_{\mathcal{C}}(S) = 0$). Moreover, due to observation (ii) above, every run of $\mathcal{A}'$ over w ultimately stays in a strongly connected component S such that $\gamma_{\mathcal{A}'}(S) = 0$. Then, by construction, there is a prefix $u \prec w$ such that $\mathcal{A}'(uw') = 0$ for all $w' \in \Sigma_{\#}^{\omega}$.

Consider a word $w \in \Sigma_{\#}^{\omega}$ such that $\mathcal{C}(w) = 1$. We want to show that $\mathit{SafetyCl}(\mathcal{A}')(w) = 1$, i.e., for every prefix $u \prec w$ we have $\mathcal{A}'(uw') = 1$ for some $w' \in \Sigma_{\#}^{\omega}$. Since $\mathcal{C}(w) = 1$, some run of $\mathcal{C}$ over w ultimately stays in a strongly connected component S such that $\gamma_{\mathcal{C}}(S) = 1$. By construction of $\mathcal{C}$, the bottom strongly connected component T_1 is reachable from any such component S. Recall that $\mathcal{A}'$ only differs from $\mathcal{C}$ in transition weights. Then, every run ρ of $\mathcal{A}'$ over w follows the same states and the strongly connected components as $\mathcal{C}$, and thus the component T_1 is reachable from any component visited during ρ by reading $\#$. Moreover, since T_1 is a bottom strongly connected component with $\gamma_C(T_1) = 1$, we have $\gamma_{\mathcal{A}'}(T_1) = 1$. Then, for every prefix $u \prec w$ we have $\mathcal{A}'(uw') = 1$ for $w' = \#^{\omega}$. □

Recall that, thanks to Theorems 5.6.21 and 5.7.2, an automaton $\mathcal{A}$ expresses a liveness property iff $\mathit{SafetyCl}(\mathcal{A})$ expresses the constant function $\top$. For automata classes whose safety closure can be expressed as Inf-automata, we provide a matching upper bound by simply checking the universality of the safety closure with respect to its top value. For DSum-automata, whose universality problem is open, our solution is based on our constant-function-check algorithm (see Theorem 5.8.3).

Theorem 5.10.2. *Deciding whether an* Inf-, Sup-, LimInf-, LimSup-, LimInfAvg-, LimSupAvg- *or* DSum-*automaton expresses a liveness property is* PSPACE*-complete.*

Proof. PSPACE-hardness is shown in Theorem 5.10.1. Let $\mathcal{A}$ be a Val-automaton and let $\top$ be its top value. Recall that liveness and top liveness coincide for sup-closed properties by Theorem 5.6.21. As the considered value functions define sup-closed properties, as proved in Theorem 5.7.2, we reduce the statement to checking whether $\mathit{SafetyCl}(\mathcal{A})$ expresses the constant function $\top$.

For Val $\in$ {Sup, LimInf, LimSup, LimInfAvg, LimSupAvg}, we first construct in PTIME an Inf-automaton $\mathcal{B}$ expressing the safety closure of $\mathcal{A}$ thanks to Theorem 5.9.6. Then, we decide in PSPACE whether $\mathcal{B}$ is equivalent to the constant function $\top$, thanks to Theorems 5.7.2 and 5.8.2 For Val = DSum, the safety closure of $\mathcal{A}$ is $\mathcal{A}$ itself, as DSum is a discounting value function due to Theorems 5.9.3 and 5.9.4. Hence, we can decide in PSPACE whether $\mathcal{A}$ expresses the constant function $\top$, thanks to Theorems 5.7.2 and 5.8.3. □

5.10.2 Safety-Liveness Decompositions of Quantitative Automata

We turn to safety-liveness decomposition, and start with the simple case of Inf- and DSum-automata, which are guaranteed to be safe. Their decomposition thus consists of only generating a liveness component, which can simply express a constant function that is at least as high as the maximal possible value of the original automaton $\mathcal{A}$. Assuming that the maximal transition weight of $\mathcal{A}$ is fixed, it can be done in constant time.

Considering Sup-automata, recall that their safety closure might not be expressible by Sup-automata (Theorem 5.9.8). Therefore, our decomposition of deterministic Sup-automata takes the safety component as an Inf-automaton. The key idea is to copy the state space of

the original automaton and manipulate the transition weights depending on how they compare with the safety-closure automaton.

Theorem 5.10.3. *Given a deterministic* Sup*-automaton $\mathcal{A}$, we can construct in* PTime *a deterministic safety* Inf*-automaton $\mathcal{B}$ and a deterministic liveness* Sup*-automaton $\mathcal{C}$, such that $\mathcal{A}(w) = \min(\mathcal{B}(w), \mathcal{C}(w))$ for every infinite word $w \in \Sigma^\omega$.*

Proof. Given a deterministic Sup-automaton, we can compute in PTime, due to Theorem 5.7.1, an equivalent deterministic Sup-automaton $\mathcal{A}$ for which every run yields a nondecreasing weight sequence. We first provide the construction of the automata $\mathcal{B}$ and $\mathcal{C}$, then show that they decompose $\mathcal{A}$, and finally prove that $\mathcal{B}$ is safe and $\mathcal{C}$ is live.

By Theorem 5.9.6, we can construct in PTime an Inf-automaton $\mathcal{B}$ expressing the safety closure of $\mathcal{A}$, where every run of $\mathcal{B}$ yields a nonincreasing weight sequence. Observe that $\mathcal{B}$ is safe by construction, and that the structures of $\mathcal{A}$ and $\mathcal{B}$ only differ on the weights appearing on transitions, where each transition weight in $\mathcal{B}$ is the maximal value that $\mathcal{A}$ can achieve after taking this transition. In particular, $\mathcal{B}$ is deterministic because $\mathcal{A}$ is so.

Then, we construct the deterministic Sup-automaton $\mathcal{C}$ by modifying the weights of $\mathcal{A}$ as follows. For every transition, if the weight of the corresponding transitions in $\mathcal{A}$ and $\mathcal{B}$ are the same, then the weight in $\mathcal{C}$ is defined as the top value of $\mathcal{A}$, denoted by $\top$ here after. Otherwise, the weight in $\mathcal{C}$ is defined as the weight of the corresponding transition in $\mathcal{A}$.

Next, we prove that $\mathcal{A}(w) = \min(\mathcal{B}(w), \mathcal{C}(w))$ for every word w. Let $\rho_\mathcal{A}, \rho_\mathcal{B}, \rho_\mathcal{C}$ be the respective runs of $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$ on w. There are the following two cases.

- If the sequences of weights $\gamma(\rho_\mathcal{A})$ and $\gamma(\rho_\mathcal{B})$ never agree, i.e., for every $i \in \mathbb{N}$ we have $\gamma(\rho_\mathcal{A}[i]) < \gamma(\rho_\mathcal{B}[i])$, then $\gamma(\rho_\mathcal{C}[i]) = \gamma(\rho_\mathcal{A}[i])$ for all $i \in \mathbb{N}$ by the construction of $\mathcal{C}$. We thus get $\mathcal{A}(w) = \mathcal{C}(w) < \mathcal{B}(w)$, so $\mathcal{A}(w) = \min(\mathcal{B}(w) < \mathcal{C}(w))$, as required.
- Otherwise, the sequences of weights $\gamma(\rho_\mathcal{A})$ and $\gamma(\rho_\mathcal{B})$ agree on at least one position, i.e., there exists $i \in \mathbb{N}$ such that $\gamma(\rho_\mathcal{A}[i]) = \gamma(\rho_\mathcal{B}[i])$. Since the run of $\mathcal{A}$ is guaranteed to yield nondecreasing weights and $\mathcal{B}$ is its safety closure, whose runs are nonincreasing, we have $\gamma(\rho_\mathcal{A}[j]) = \gamma(\rho_\mathcal{B}[j])$ for all $j \geq i$. Additionally, $\gamma(\rho_\mathcal{C}[i]) = \top$ by the construction of $\mathcal{C}$. We thus get $\mathcal{A}(w) = B(w) < \mathcal{C}(w)$, so $\mathcal{A}(w) = \min(\mathcal{B}(w) < \mathcal{C}(w))$, as required.

Finally, we show that $\mathcal{C}$ is live. By Theorem 5.6.21, it is sufficient to show that for every reachable state q of $\mathcal{C}$, there exists a run starting from q that visits a transition weighted by $\top$. Suppose towards contradiction that for some state $\hat{q}$, there is no such run. Recall that the state spaces and transitions of $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$ are the same. Moreover, observe that a transition weight in $\mathcal{C}$ is $\top$ if and only if the corresponding transitions in $\mathcal{A}$ and $\mathcal{B}$ have the same weight.

If no transition with weight $\top$ is reachable from the state $\hat{q}$, then by the construction of $\mathcal{C}$, for every run $\rho_\mathcal{A}$ of $\mathcal{A}$ starting from $\hat{q}$ and the corresponding run $\rho_\mathcal{B}$ of $\mathcal{B}$, we have $\gamma(\rho_\mathcal{A}[i]) < \gamma(\rho_\mathcal{B}[i])$ for all $i \in \mathbb{N}$. Recall that each transition weight in $\mathcal{B}$ is the maximal value $\mathcal{A}$ can achieve after taking this transition, and that for every finite word u over which $\mathcal{A}$ reaches $\hat{q}$, we have $\sup_{w'} \mathcal{A}(uw') = \mathcal{B}(uw')$.

Hence, by the sup-closedness of $\mathcal{A}$ and the fact that the sequences of weights in its runs are nondecreasing, for each prefix $r_\mathcal{A}$ of $\rho_\mathcal{A}$ and the corresponding prefix $r_\mathcal{B}$ of $\rho_\mathcal{B}$, there is an infinite continuation $\rho'_\mathcal{A}$ for $r_\mathcal{A}$ such that the corresponding infinite continuation $\rho'_\mathcal{B}$ for $r_\mathcal{B}$ gives $\mathsf{Sup}(\gamma(r_\mathcal{A}\rho'_\mathcal{A})) = \mathsf{Inf}(\gamma(r_\mathcal{B}\rho'_\mathcal{B}))$. Note that this holds only if the two weight sequences have

the same value after some finite prefix, in which case the weight of $\mathcal{C}$ is defined as $\top$. Hence, some run of $\mathcal{C}$ from $\hat{q}$ reaches a transition weighted $\top$, which yields a contradiction. □

Using the same idea, but with a more involved reasoning, we show a safety-liveness decomposition for deterministic LimInf- and LimSup-automata.

Theorem 5.10.4. *Let* Val $\in \{\text{LimInf}, \text{LimSup}\}$. *Given a deterministic* Val*-automaton $\mathcal{A}$, we can construct in* PTIME *a deterministic safety* Val*-automaton $\mathcal{B}$ and a deterministic liveness* Val*-automaton $\mathcal{C}$, such that $\mathcal{A}(w) = \min(\mathcal{B}(w), \mathcal{C}(w))$ for every infinite word $w \in \Sigma^\omega$.*

Proof. Consider a deterministic Val-automaton $\mathcal{A}$. We construct $\mathcal{B}$ and $\mathcal{C}$ analogously to their construction in the proof of Theorem 5.10.3, with the only difference that we use Theorem 5.9.7 to construct $\mathcal{B}$ as a Val-automaton rather than an Inf-automaton. Once again, the structures of $\mathcal{A}$ and $\mathcal{B}$ only differ on the weights appearing on transitions, and $\mathcal{B}$ is deterministic because $\mathcal{A}$ is so.

We first show that $\mathcal{B}$ and $\mathcal{C}$ decompose $\mathcal{A}$, and then prove that $\mathcal{C}$ is live. (Note that $\mathcal{B}$ is safe by construction.)

Given an infinite word w, let $\rho_\mathcal{A}, \rho_\mathcal{B}, \rho_\mathcal{C}$ be the respective runs of $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$ on w. There are the following three cases.

- If the sequences of weights $\gamma(\rho_\mathcal{A})$ and $\gamma(\rho_\mathcal{B})$ agree only on finitely many positions, i.e., there exists $i \in \mathbb{N}$ such that $\gamma(\rho_\mathcal{A}[j]) < \gamma(\rho_\mathcal{B}[j])$ for all $j \geq i$, then by the construction of $\mathcal{C}$, we have $\gamma(\rho_\mathcal{C}[j]) = \gamma(\rho_\mathcal{A}[j])$ for all $j \geq i$. Thus, $\mathcal{A}(w) = \mathcal{C}(w) < \mathcal{B}(w)$.
- If the sequences of weights $\gamma(\rho_\mathcal{A})$ and $\gamma(\rho_\mathcal{B})$ disagree only on finitely many positions, i.e., there exists $i \in \mathbb{N}$ such that $\gamma(\rho_\mathcal{A}[j]) = \gamma(\rho_\mathcal{B}[j])$ for all $j \geq i$, then by the construction of $\mathcal{C}$, we have $\gamma(\rho_\mathcal{C}[j]) = \top$ for all $j \geq i$. Thus, $\mathcal{A}(w) = \mathcal{B}(w) \leq \mathcal{C}(w)$.
- Otherwise the sequences of weights $\gamma(\rho_\mathcal{A})$ and $\gamma(\rho_\mathcal{B})$ both agree and disagree on infinitely many positions, i.e., for every $i \in \mathbb{N}$ there exist $j, k \geq i$ such that $\gamma(\rho_A[j]) < \gamma(\rho_\mathcal{B}[j])$ and $\gamma(\rho_A[k]) = \gamma(\rho_\mathcal{B}[k])$. For Val = LimInf, we exhibit an infinite sequence of positions $\{x_i\}_{i\in\mathbb{N}}$ such that $\gamma(\rho_A[x_i]) = \gamma(\rho_\mathcal{C}[x_i]) < \gamma(\rho_\mathcal{B}[x_i])$ for all $i \in \mathbb{N}$. The first consequence is that $\mathcal{A}(w) < \mathcal{B}(w)$. The second consequence is that, by the construction of $\mathcal{C}$, if $\mathcal{A}(w) < \top$ then $\mathcal{C}(w) < \top$, which implies that $\mathcal{A}(w) = \mathcal{C}(w)$. For Val = LimSup, recall that every run of $\mathcal{B}$ yields a nonincreasing weight sequence. In particular, there exists $k \in \mathbb{N}$ such that $\gamma(\rho_\mathcal{B}[k]) = \gamma(\rho_\mathcal{B}(\ell)) = \mathcal{B}(w)$ for all $\ell \geq k$. Then, we exhibit an infinite sequence of positions $\{y_i\}_{i\in\mathbb{N}}$ such that $\gamma(\rho_A[y_i]) = \mathcal{B}(\gamma(\rho_\mathcal{B}[y_i])) = \mathcal{B}(w)$ and $\gamma(\rho_\mathcal{C}[y_i]) = \top$ for all $i \in \mathbb{N}$. Consequently, $\mathcal{C}(w) = \top$ and $\mathcal{A}(w) = \mathcal{B}(w)$.

In either case, $\mathcal{A}(w) = \min(\mathcal{B}(w), \mathcal{C}(w))$.

Next, we show that $\mathcal{C}$ is live using the same argument as in the proof of Theorem 5.10.3: On the one hand, every word w for which $\mathcal{A}(w) = \mathcal{B}(w)$ trivially satisfies the liveness condition as it implies $\mathcal{C}(w) = \top$. On the other hand, by Theorem 5.7.2 every word w for which $\mathcal{A}(w) < \mathcal{B}(w)$ is such that each finite prefix $u \prec w$ admits a continuation w' satisfying $\mathcal{A}(uw') = \mathcal{B}(uw')$. Hence, $\sup_{w'} \mathcal{C}(uw') = \top$ for all $u \prec w$, implying the liveness condition. □

Finally, we provide a safety-liveness decomposition for nondeterministic automata with the prefix-independent value functions we consider.

Theorem 5.10.5. *Let* Val $\in \{\text{LimSup}, \text{LimInf}, \text{LimInfAvg}, \text{LimSupAvg}\}$. *Given a* Val*-automaton* $\mathcal{A}$*, we can construct in* PTime *a safety* Val*-automaton* $\mathcal{B}$ *and a liveness* Val*-automaton* $\mathcal{C}$*, such that* $\mathcal{A}(w) = \min(\mathcal{B}(w), \mathcal{C}(w))$ *for every infinite word* $w \in \Sigma^\omega$.

Proof. Let $Q = \{q_1, \ldots, q_n\}$ be the set of states of $\mathcal{A}$, let $\Delta_\mathcal{A}$ be its transition relation, $\gamma_\mathcal{A}$ its weight function, and $X_\mathcal{A}$ its finite set of weights. We identify in PTime the strongly connected components $Q_1 \uplus Q_2 \uplus \ldots \uplus Q_m$ of $\mathcal{A}$. For all $k \in \{1, \ldots, m\}$, we compute in PTime, thanks to Theorem 5.7.2, the top value $\top_k$ of the automaton $\mathcal{A}^q$ for any $q \in Q_k$. Note that the choice of $q \in Q_k$ does not change $\top_k$ since the considered value function Val is prefix independent. Additionally, for all $k \in \{1, \ldots, m\}$, we compute the highest value Θ_k achievable by some simple cycle π_k within Q_k. To clarify, we emphasize that $\top_k \geq \Theta_k$ holds in general, and $\top_k > \Theta_k$ when all runs starting in Q_k that achieve the top value $\top_k$ eventually leave the component Q_k.

We explain briefly how Θ_k and π_k are computed in PTime. The value Θ_k is the top value of the automaton consisting of Q_k and a sink absorbing all outgoing edges weighted with $\min X_\mathcal{A} - 1$. As discussed in the proof of [CDH10b, Thm. 3], the top value of a Val-automaton is attainable by a lasso run. Due to the properties of Val, this lasso run can be transformed into a simple cycle, i.e., a cycle without inner cycles. Because Val is prefix independent, the path reaching the cycle of the lasso run can be removed to obtain a cycle run with the same value. Also, if the cycle $\rho = \rho_1\rho_2\rho_3$ has an inner cycle ρ_2, then ρ can be shortened by keeping the cycle achieving the highest value between ρ_2 and $\rho_1\rho_3$. This proves that Θ_k is attainable by a simple-cycle run.

Now, we briefly describe the computation of Θ_k and π_k. First, we consider Val $\in \{\text{LimInf}, \text{LimSup}\}$. To compute Θ_k, we first construct a Val-automaton $\mathcal{A}_k$ that is a copy of the strongly connected component Q_k extended to be total by adding a sink state with a self loop of weight $\min X_\mathcal{A} - 1$. Then, we compute the top value of $\mathcal{A}_k$, which is by definition Θ_k. To compute π_k, we first construct a graph G_k which is obtained from the underlying graph of $\mathcal{A}_k$ by removing all the edges corresponding to transitions of $\mathcal{A}_k$ whose weights are smaller than Θ_k if Val = LimInf or greater than Θ_k if Val = LimSup. Then, we compute a cycle in G_k using depth-first search and assign it to π_k.

Second, we consider Val $\in \{\text{LimInfAvg}, \text{LimSupAvg}\}$. To compute Θ_k, we first construct $\mathcal{A}_k$ as above, take the underlying directed graph of $\mathcal{A}_k$, and multiply its edge weights by -1. Then, we use Karp's (dynamic programming) algorithm [Kar78] to compute the minimum cycle mean in this directed graph, which gives us the value $-\Theta_k$. To compute π_k, it suffices to appropriately maintain the backtracking pointers in Karp's algorithm [CM17]. Recall that the top value of an automaton can be computed in PTime thanks to Theorem 5.7.2, and note that the constructions described above are also in PTime.

We define the set of states of $\mathcal{C}$ as $P = \{p_1, p'_1, \ldots, p_n, p'_n, p_\perp\}$, in particular $|P| = 2|Q| + 1$. In the following, we define the transition relation $\Delta_\mathcal{C}$ of $\mathcal{C}$. The states $\{p_i \mid 1 \leq i \leq n\}$ are used to copy $\mathcal{A}$, i.e., $(p_i, \sigma, p_j) \in \Delta_\mathcal{C}$ if and only if $(q_i, \sigma, q_j) \in \Delta_\mathcal{A}$. Additionally, for all $k \in \{1, \ldots, m\}$, if $\top_k = \Theta_k$ then for all transitions of the simple cycle π_k of the form $(q_i, \sigma, q_j) \in \Delta_\mathcal{A}$, we have $(p'_i, \sigma, p'_j) \in \Delta_\mathcal{C}$ and $(p_i, \sigma, p'_j) \in \Delta_\mathcal{C}$. Finally, for all p'_i and σ, we have $(p'_i, \sigma, p_\perp) \in \Delta_\mathcal{C}$ and $(p_\perp, \sigma, p_\perp) \in \Delta_\mathcal{C}$. Now, we define the weight function $\gamma_\mathcal{C}$ of $\mathcal{C}$. For all transitions of the form $t = (p_i, \sigma, p_j) \in \Delta_\mathcal{C}$, we have $\gamma_\mathcal{C}(t) = \gamma_\mathcal{A}(q_i, \sigma, q_j)$. For all transitions of the from $t = (p, \sigma, p')$ with $p \in P \setminus \{p_\perp\}$ and $p' \in \{p'_i \mid 1 \leq i \leq n\}$, we have $\gamma_\mathcal{C}(t) = \top_\mathcal{A}$. Finally, $\gamma_\mathcal{C}(p_\perp, \sigma, p_\perp) = \min X_\mathcal{A}$ for all σ. An example is given in Figure 5.3.

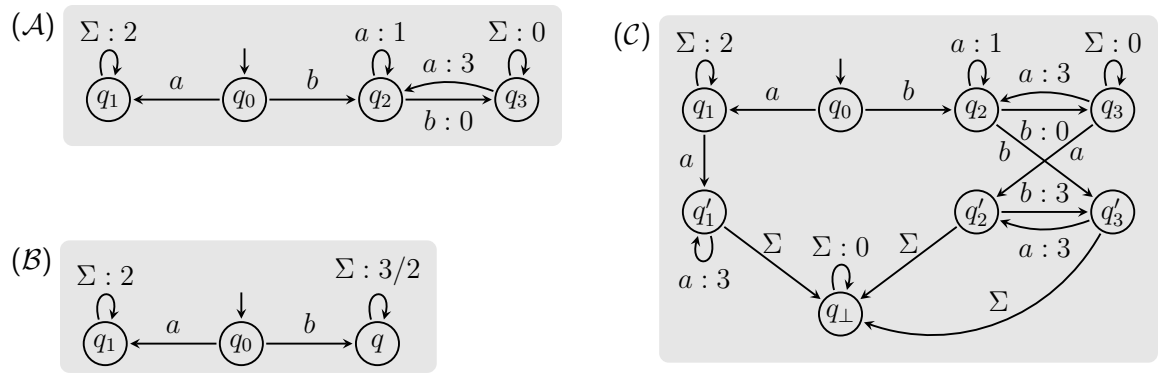


Figure 5.3: A nondeterministic LimInfAvg-automaton $\mathcal{A}$ and its safety-liveness decomposition into LimInfAvg-automata $\mathcal{B}$ and $\mathcal{C}$, as presented in the proof of Theorem 5.10.5.

Next, we prove that $\mathcal{C}$ is live. The key argument is that, for each component Q_k for which $\top_k = \Theta_k$, the automaton $\mathcal{C}$ provides a continuation leading to achieve the highest weight of $\mathcal{A}$. Recall that liveness and top liveness coincide for $\sup$-closed properties by Theorem 5.6.21. As the considered value function Val defines $\sup$-closed properties, as proved in Theorem 5.7.2, the liveness of $\mathcal{C}$ reduces to checking whether $SafetyCl(\mathcal{C})$ expresses the constant function $\top_{\mathcal{A}}$. In fact, by construction, all finite runs ending in $P \setminus \{p_\perp\}$ admit a continuation leading to achieve $\top_{\mathcal{A}}$. Additionally, for all finite runs ending in $p_\perp$, there is another run over the same word that follows the states of $\mathcal{A}$. Hence, the safety closure of $\mathcal{C}$ maps every words to $\top_{\mathcal{A}}$, implying the liveness of $\mathcal{C}$.

By Theorem 5.9.7, we can construct in PTIME a Val-automaton $\mathcal{B}$ expressing the safety closure of $\mathcal{A}$. We prove that the automata $\mathcal{B}$ and $\mathcal{C}$ yield a safety-liveness decomposition of $\mathcal{A}$. For all $w \in \Sigma^\omega$, if there is a run of $\mathcal{A}$ over w of the form $\pi\pi_k^\omega$ for some finite run π in $\mathcal{A}$, then $\top_k = \mathcal{B}(w) = \mathcal{A}(w) \leq \mathcal{C}(w) = \top_{\mathcal{A}}$, otherwise $\mathcal{A}(w) = \mathcal{C}(w)$. Since $\mathcal{A}(w) \leq \mathcal{B}(w)$ by construction, we have $\mathcal{A}(w) = \min(\mathcal{B}(w), \mathcal{C}(w))$, for all $w \in \Sigma^\omega$.

Finally, let us note that the liveness component $\mathcal{C}$ constructed here may differ from the liveness component Ψ of the decomposition in Theorem 5.6.13. To construct $\mathcal{C}$ efficiently, we only take into account one simple cycle π_k that achieves the value Θ_k within each strongly connected component S_k. However, there may be many cycles within S_k achieving Θ_k, which would need to be taken into account to express Ψ. □

Nondeterministic Sup-automata can be handled as LimInf- or LimSup-automata (Theorem 5.7.1) and decomposed accordingly. For deterministic automata, the decomposition in Theorem 5.10.5 yields a deterministic safety component, but its liveness component may be nondeterminizable. Whether deterministic LimInfAvg- and LimSupAvg-automata can be decomposed into deterministic automata remains open.

5.11 Conclusion

We presented a generalization of safety and liveness that lifts the safety-progress hierarchy to the quantitative setting of [CDH10b] while preserving major desirable features of the boolean setting such as the safety-liveness decomposition and connections to topology. Then, we instantiated our framework with the specific classes of quantitative properties expressed by automata.

Monitorability identifies a boundary separating properties that can be verified or falsified from a finite number of observations, from those that cannot. Safety-liveness and co-safety-co-liveness decompositions allow us separate, for an individual property, monitorable parts from nonmonitorable parts. The larger the monitorable parts of the given property, the stronger the decomposition. We provided the strongest known safety-liveness decomposition, which consists of a pointwise minimum between a safe part defined by a quantitative safety closure, and a live part which corrects for the difference.

Moreover, we studied the quantitative safety-liveness dichotomy for properties expressed by Inf-, Sup-, LimInf-, LimSup-, LimInfAvg-, LimSupAvg-, and DSum-automata. To this end, and solved the constant-function problem for these classes of automata. We presented automata-theoretic constructions for the safety closure of these automata and decision procedures for checking their safety and liveness. We proved that the value function Inf yields a class of safe automata and DSum both safe and co-safe. For all common automata classes, we provided a decomposition into a safe and a live component. We emphasize that the safety component of our decomposition algorithm is the safety closure, and thus the best safe approximation of a given automaton. We note that most of these algorithms have been recently implemented in a tool [CHMS24, CHMS25].

We focused on quantitative automata [CDH10b] because their totally-ordered value domain and their $\sup$-closedness make quantitative safety and liveness behave in particularly natural ways; a corresponding investigation of weighted automata [Sch61] remains to be done. We left open the complexity gap in the safety check of limit-average automata, and the study of co-safety and co-liveness for nondeterministic quantitative automata, which is not symmetric to safety and liveness due to the nonsymmetry in resolving nondeterminism by the supremum value of all possible runs.

CHAPTER 6

QuAK: Quantitative Automata Kit

In this chapter, the following publications were re-used in full:

- Marek Chalupa, Thomas A. Henzinger, Nicolas Mazzocchi, N. Ege Saraç. *QuAK: Quantitative Automata Kit*. In Leveraging Applications of Formal Methods, Verification and Validation. Software Engineering Methodologies - 12th International Symposium, **ISoLA 2024**.
- Marek Chalupa, Thomas A. Henzinger, Nicolas Mazzocchi, N. Ege Saraç. *Automating the Analysis of Quantitative Automata with QuAK*. In Tools and Algorithms for the Construction and Analysis of Systems - 31st International Conference, **TACAS 2025**.

6.1 Introduction

System behaviors are traditionally seen as sequences of system events, and specifications typically categorize them as correct or incorrect without providing more detailed information. This binary perspective has long been the cornerstone of formal verification. However, many interesting system properties require moving beyond this view to systematically reason about timing constraints, uncertainty, resource consumption, robustness, and more, which necessitates a more nuanced approach to the specification, modeling, and analysis of computer systems.

Quantitative automata [CDH10b] extend standard boolean ω-automata with weighted transitions and a value function that accumulates an infinite sequence of weights into a single value, which generalizes the notion of acceptance condition. The common value functions include Inf, Sup, LimInf, and LimSup (respectively generalizing safety, reachability, co-Büchi and Büchi acceptance conditions), as well as DSum (discounted sum), LimInfAvg and LimSupAvg (limit average a.k.a. mean payoff). Let us consider the quantitative automaton $\mathcal{A}$ given in Figure 6.1, which models the power consumption of a device. With the value function Inf, it maps each execution to its minimal power consumption, whereas with LimInfAvg or LimSupAvg to its long-term average power consumption. For example, the infinite execution $(\text{off} \cdot \text{on})^\omega$ is mapped to 0 with the value function Inf, and to 1 with LimInfAvg or LimSupAvg.

The decision problems for boolean automata extend naturally to the quantitative setting. A quantitative automaton $\mathcal{A}$ is defined to be nonempty (resp. universal) with respect to a rational threshold v if it maps some (resp. every) infinite word w to a value at

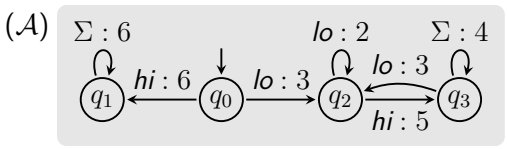


Figure 6.1: A nondeterministic automaton $\mathcal{A}$ over the alphabet $\Sigma = \{hi, lo\}$, modeling the power consumption of a device where starting with high-power mode is not reversible. Associating with $\mathcal{A}$ different value functions, we can specify different aspects of its power consumption, e.g., considering LimInfAvg, the automaton specifies the long-term average power consumption.

least v [CDH10b]. Such problems are closely related to computing the extremal values (top or bottom) of an automaton and have significant implications both in theory and in the practical verification of systems. Over the past fifteen years, quantitative automata have been intensively studied [BH14, BHO15, MO19, MO21, Bok24], with recent work focusing on monitorability as well as safety and liveness properties [HS21, HMS22, HMS23, BHMS23].

Despite these theoretical advances, no general-purpose tool for the analysis of quantitative automata existed until Quantitative Automata Kit (QuAK). Our tool QuAK supports a wide range of automaton types—including Inf, Sup, LimInf, LimSup, LimInfAvg, and LimSupAvg—and implements decision procedures for fundamental problems such as nonemptiness, universality, inclusion, equivalence, constant-check, safety, and liveness.

Related work Modeling beyond-boolean aspects of systems has been considered in several different ways. One approach considers multi-valued truth domains instead of binary domains [BG99, CGD02]. Another prominent approach involves weighted automata [Sch61], which extend classical automata by assigning each transition a numerical weight from a semiring whose operations describe how the weights are accumulated. Tools such as Vaucanson [LPRS03], Vcsn [DDLS13], and Awali [LMS22] provide support for the analysis of weighted automata. The well-established techniques for weighted automata on finite words do not adapt well to the ω-valuation monoid framework necessary for infinite words. Quantitative automata provide a more intuitive alternative, as they are designed to generalize boolean finite-state ω-automata. See [Bok21] for more on the distinction between weighted and quantitative automata. Another significant approach considers the interaction of digital computational processes with analog physical processes, modeled using automata [AD94, Hen96] as well as temporal logics [AH93] and implemented in tools such as UPPAAL [LPY97] and HyTech [HH94]. Signal temporal logic [MN04], in particular, has quantitative semantics that allows for reasoning about the degree to which a specification is satisfied or violated, and is implemented in tools such as Breach [Don10], S-TaLiRo [ALFS11], and RTAMT [NY20, YHN24]. Finally, probabilistic verification deals with systems that have inherent uncertainties, such as random failures or probabilistic decision making. PRISM [KNP02] and STORM [DJKV17] are widely used tools that allows for the analysis of probabilistic models like Markov chains and Markov decision processes.

6.2 Quantitative Automata

In this section we recall quantitative automata and their decision problems. For formal definitions, we refer the reader to Chapters 2 and 5.

While quantitative properties define total functions from infinite words to a complete lattice, quantitative automata define a subset of quantitative properties on totally-ordered value domains. Informally, quantitative automata are finite-state automata with weighted edges, where the edge weights come from a finite subset of $\mathbb{Q}$. Note that we require quantitative automata to be total (a.k.a. complete), i.e., there is an outgoing transition from every state with each letter. The semantics of quantitative automata is largely determined by their value function, a function from infinite sequences of rational weights $\mathbb{Q}^\omega$ to real numbers $\mathbb{R}$, which generalize the acceptance conditions of boolean automata. We consider the below value functions over an infinite sequence $x = x_0 x_1 \dots$ of rational weights.

- $\mathsf{Inf}(x) = \inf\{x_n \mid n \geq 0\}$
- $\mathsf{Sup}(x) = \sup\{x_n \mid n \geq 0\}$
- $\mathsf{LimInf}(x) = \lim_{n\to\infty} \inf\{x_i \mid i \geq n\}$
- $\mathsf{LimSup}(x) = \lim_{n\to\infty} \sup\{x_i \mid i \geq n\}$
- $\mathsf{LimInfAvg}(x) = \mathsf{LimInf}\left(\frac{1}{n}\sum_{i=0}^{n-1} x_i\right)$
- $\mathsf{LimSupAvg}(x) = \mathsf{LimSup}\left(\frac{1}{n}\sum_{i=0}^{n-1} x_i\right)$
- For a discount factor $\lambda \in \mathbb{Q} \cap (0,1)$, $\mathsf{DSum}_\lambda(x) = \sum_{i\geq 0} \lambda^i x_i$

Let us consider the automaton $\mathcal{A}$ given in Figure 6.1. Suppose we pair it with the value function LimSup. The word $w = \mathit{lo}^\omega$ yields a unique run with the weight sequence $x = 3, 2, 2, 2, \dots$ for which we have $\mathsf{LimSup}(x) = 2$, therefore $\mathcal{A}(w) = 2$. When a word yields multiple runs (i.e., the automaton is nondeterministic) we resolve the nondeterminism by taking the supremum over the values obtained from these runs. For example, if $w = \mathit{lo} \cdot \mathit{hi} \cdot \mathit{lo}^\omega$, the automaton has infinitely many runs over w. One of these runs stay and loop at q_3 indefinitely, resulting in a weight sequence whose tail is 4^ω, therefore $\mathcal{A}(w) = 4$.

The *top value* of an automaton $\mathcal{A}$ is $\top_\mathcal{A} = \sup_{w\in\Sigma^\omega} \mathcal{A}(w)$, and its *bottom value* is $\bot_\mathcal{A} = \inf_{w\in\Sigma^\omega} \mathcal{A}(w)$.

Quantitative Automata Problems

We describe the standard decision problems of quantitative automata as well as the problems related to their safety and liveness. The complexity results are summarized in Table 6.1.

An automaton $\mathcal{A}$ is *nonempty* (resp. *universal*) with respect to a threshold $v \in \mathbb{Q}$ iff $\mathcal{A}(w) \geq v$ for some (resp. all) $w \in \Sigma^\omega$. Nonemptiness (resp. universality) is closely related to computing an automaton's top value (resp. bottom value): $\mathcal{A}$ is nonempty (resp. universal) with respect to $v \in \mathbb{Q}$ iff $\top_\mathcal{A} \geq v$ (resp. $\bot_\mathcal{A} \geq v$). An automaton $\mathcal{A}$ is *included in* (resp. *equivalent to*) an automaton $\mathcal{B}$ iff $\mathcal{A}(w) \leq \mathcal{B}(w)$ (resp. $\mathcal{A}(w) = \mathcal{B}(w)$) for all $w \in \Sigma^\omega$. An automaton $\mathcal{A}$ is *constant* iff there exists $c \in \mathbb{R}$ such that $\mathcal{A}(w) = c$ for all $w \in \Sigma^\omega$. This problem is closely related to safety and liveness of quantitative automata, as we discuss below.

Quantitative safety generalizes the boolean view by considering membership hypotheses in the form of lower bound queries: a property is safe iff every wrong membership hypothesis has a finite witness for the violation. Formally, a quantitative property $\Phi : \Sigma^\omega \to \mathbb{D}$ is *safe* iff for every $w \in \Sigma^\omega$ and $v \in \mathbb{D}$ with $\Phi(w) \not\geq v$, there exists a finite prefix $u \prec w$ such that $\sup_{w'\in\Sigma^\omega} \Phi(uw') \not\geq v$ [HMS23]. Moreover, an automaton $\mathcal{A}$ is safe iff the quantitative property defined by $\mathcal{A}$ is safe. Given a quantitative property $\Phi : \Sigma^\omega \to \mathbb{D}$, its *safety closure* is defined as $\mathit{SafetyCl}(\Phi)(w) = \inf_{u\prec w} \sup_{w'\in\Sigma^\omega} \Phi(uw')$ and is the least safety property that

	Inf	Sup, LimInf, LimSup	LimInfAvg, LimSupAvg	DSum
Nonemptiness check	PTime			
Universality check	PSpace-complete		Undecidable	Open
Inclusion check	PSpace-complete		Undecidable	Open
Equivalence check	PSpace-complete		Undecidable	Open
Top value computation	PTime			
Safety closure construction	$O(1)$	PTime		$O(1)$
Safety-liveness decomposition	$O(1)$	PTime		$O(1)$
Safety check	$O(1)$	PSpace-complete	ExpSpace; PSpace-hard	$O(1)$
Liveness check	PSpace-complete			
Constant-function check	PSpace-complete			

Table 6.1: The complexity of performing the operations on the left column with respect to nondeterministic automata with the value function on the top row. The decidability results in the top five rows are shown in [KL07, CDH10b] and undecidability in [DDG+10, CDE+10, HPPR18]. All the results in the bottom five rows are shown in [BHMS25]. All the operations are computable in PTime for deterministic automata.

bounds Φ from above [HMS23]. As expected, a property Φ is safe iff $\Phi(w) = \mathit{SafetyCl}(\Phi)(w)$ for all $w \in \Sigma^\omega$, and we can compute the safety closure of an automaton $\mathcal{A}$—the automaton $\mathit{SafetyCl}(\mathcal{A})$ that expresses the safety closure of the property defined by $\mathcal{A}$. While this characterization is useful for some classes of quantitative automata, the equivalence problem is undecidable for LimInfAvg and LimSupAvg automata. For these, the safety problem is still decidable by a reduction to their constant-function problem [BHMS23].

Quantitative liveness extends the membership-based view: a quantitative property $\Phi : \Sigma^\omega \to \mathbb{D}$ is live iff for every word (whose value is less than $\top = \sup \mathbb{D}$) there exists a wrong membership hypothesis without a finite witness for the violation. Formally, a quantitative property $\Phi : \Sigma^\omega \to \mathbb{D}$ is *live* iff for all $w \in \Sigma^\omega$, if $\Phi(w) < \top$, then there exists a value $v \in \mathbb{D}$ such that $\Phi(w) \not\geq v$ and for all prefixes $u \prec w$, we have $\sup_{w' \in \Sigma^\omega} \Phi(uw') \geq v$ [HMS23]. Moreover, an automaton $\mathcal{A}$ is live iff the quantitative property defined by $\mathcal{A}$ is live. For the common classes of quantitative automata, deciding liveness reduces to the constant-function problem: an automaton $\mathcal{A}$ is live iff $\mathit{SafetyCl}(\mathcal{A})$ is constant [BHMS23]. Just like every boolean property is the intersection of its safety closure and a liveness property, every quantitative property is the pointwise minimum of its safety closure and a liveness property [HMS23]. Recently, it was proved that all the common classes of automata can be decomposed into its safety closure and a liveness property [BHMS25]. Consider the automaton $\mathcal{A}$, its safety closure $\mathcal{B}$, and its liveness part $\mathcal{C}$ as defined in Figure 6.2. In $\mathcal{B}$, each strongly connected component (SCC) of $\mathcal{A}$ is assigned the highest value achievable within the component, representing the greatest among the lower bound hypotheses that cannot be refuted by any finite prefix. The liveness part $\mathcal{C}$ consists of three components: the upper part is a copy of $\mathcal{A}$ (ensuring $\mathcal{C}$ can have runs with the same value as $\mathcal{A}$); the middle part contains a $\top_{\mathcal{A}}$-weighted copy of the highest-valued cycle in each SCC (enabling $\mathcal{C}$ to achieve high-valued runs when $\mathcal{A}$ and $\mathcal{B}$ agree); and the lower part includes a sink state looping with the lowest weight of $\mathcal{A}$ (allowing $\mathcal{C}$ to "escape" the middle part and realize a value using the upper part).

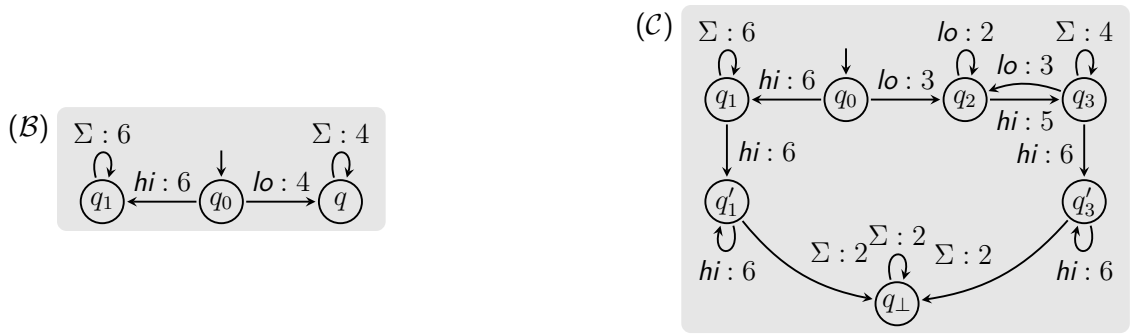


Figure 6.2: Taking the automaton $\mathcal{A}$ in Figure 6.1 as a LimInfAvg automaton, the automaton $\mathcal{B}$ denotes the safety closure of $\mathcal{A}$ and the automaton $\mathcal{C}$ its liveness component in the corresponding decomposition [BHMS25].

6.3 The Tool

Let $\mathcal{A}$ and $\mathcal{B}$ be Val automata where Val $\in \{\mathsf{Inf}, \mathsf{Sup}, \mathsf{LimInf}, \mathsf{LimSup}, \mathsf{LimInfAvg}, \mathsf{LimSupAvg}\}$, and let $v \in \mathbb{Q}$ be a rational number. QuAK currently supports the following operations (whenever known to be computable):

1. Check if $\mathcal{A}$ is non-empty with respect to v.
2. Check if $\mathcal{A}$ is universal with respect to v.
3. Check if $\mathcal{A}$ is included in $\mathcal{B}$.
4. Check if $\mathcal{A}$ defines a constant function.
5. Check if $\mathcal{A}$ defines a safety property.
6. Check if $\mathcal{A}$ defines a liveness property.
7. Compute the top value $\top$ of $\mathcal{A}$.
8. Compute the bottom value $\bot$ of $\mathcal{A}$.
9. Compute the safety closure of $\mathcal{A}$.
10. Compute the safety-liveness decomposition of $\mathcal{A}$.
11. Construct and execute a monitor for $\mathcal{A}$.

The tool is written in C++ using the standard library, and is available at `https://github.com/ista-vamos/QuAK` together with detailed instructions on its usage. In this section, we describe the automata representation, the architecture of QuAK, our antichain-based inclusion algorithm, the witness computation for the supported operations, implementation of the constant-function check for limit average automata, and monitoring approach for quantitative properties.

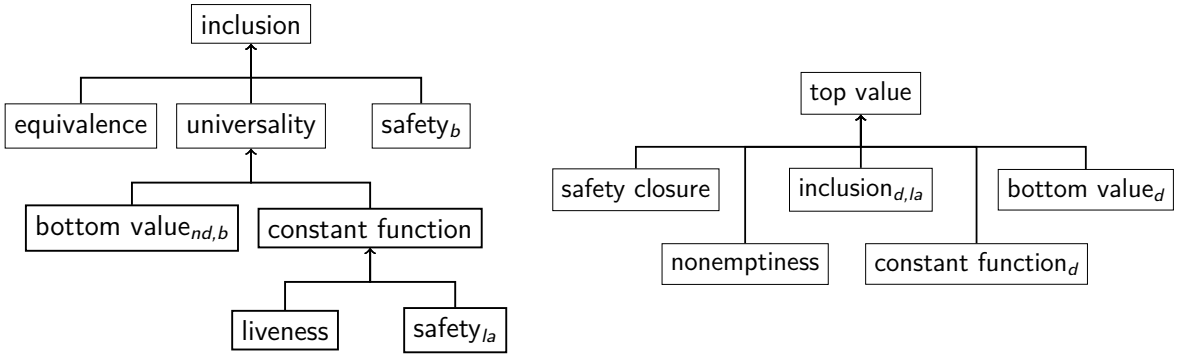


Figure 6.3: Reductions of quantitative automata problems in QuAK. The subscript *b* stands for basic (i.e., Val $\in$ {Inf, Sup, LimInf, LimSup}), *la* for limit-average (i.e., Val $\in$ {LimInfAvg, LimSupAvg}), *d* for deterministic, and *nd* for nondeterministic.

6.3.1 Automata Representation

We represent automata in a way that makes the algorithms as efficient as possible while keeping their implementation convenient and maintainable. Here, we explain some choices we made for this sake.

Automata objects do not have a value function because it may be useful to interpret the same transition structure in different ways (like in Figure 6.1). The user needs to specify the value function when a decision procedure or a construction is called on an automaton. Moreover, each automaton contains a directed acyclic graph representing its strongly connected components (SCCs), which is constructed once the automaton is created. Each state has a tag representing the SCC it belongs to. Moreover, in addition to storing the outgoing transitions of a state, we also store the incoming transitions. These are useful when computing the top value, constructing the safety closure, and determinizing the safety closure of limit average automata (for deciding their safety). Finally, while boolean automata has a fixed domain $\{0, 1\}$, each quantitative automaton may define a different domain. To address this, each automaton has two numerical variables representing the minimum and maximum of its domain, whose values are taken as the minimum and maximum of the automaton's weights by default.

6.3.2 Structure of QuAK

Figure 6.3 illustrates the overall reduction strategy implemented in QuAK. At its core, QuAK is built around two fundamental algorithms: the inclusion check and the top value computation. These two procedures serve as the basis for solving all other decision problems in quantitative automata, helping us achieve a modular and efficient design.

In the left subfigure, the inclusion algorithm is shown at the root of a reduction tree. Other problems such as equivalence and universality reduce to inclusion. For example, for nondeterministic automata with the "basic" value functions Inf, Sup, LimInf, LimSup (denoted by the subscript *nd,b*), the bottom value is computed by a reduction to the universality problem: the greatest weight for which the automaton is universal gives us the bottom value. In turn, these universality checks are treated as instances of the inclusion problem: an automaton $\mathcal{A}$ is universal w.r.t v iff it includes the single-state automaton that maps all words to v.

The right subfigure shows the role of the top value algorithm. For example, the bottom value of a deterministic automaton $\mathcal{A}$ with the value function Val is computed as the top value of

the copy $\mathcal{A}'$ of $\mathcal{A}$ where all weights are multiplied by -1 and the value function replaced by its dual (e.g., $\mathcal{A}'$ is an Inf-automaton if Val = Sup).

6.3.3 Antichain-based Inclusion Algorithm

The language inclusion problem of Büchi automata is known to be PSPACE-complete. Algorithms that behave well in practice have been investigated for decades and remain an active research field. Among others, FORKLIFT [DGM22a] uses the Ramsey-based technique and leverages the antichain heuristic. The algorithm to decide whether $L(\mathcal{A}) \subseteq L(\mathcal{B})$ holds searches for counterexamples (i.e., words that are in $L(\mathcal{A})$ but not $L(\mathcal{B})$) by systematically running membership queries. The Ramsey-based approach prunes the search for counterexamples by discarding candidate words in $L(\mathcal{A})$ which are "subsumed" by other words of $L(\mathcal{A})$ with respect to a given well-quasiorder. Termination comes from the mathematical properties of well-quasiorders guaranteeing that only finitely many candidates will be kept after pruning. Correctness is trivial: if a kept candidate witnesses the violation of $L(\mathcal{A}) \subseteq L(\mathcal{B})$ then the inclusion does not hold. To guarantee completeness, we require the quasiorder to fulfill, for all candidates $w \in \Sigma^\omega$ subsumed by $w_0 \in \Sigma^\omega$, that $w_0 \in L(\mathcal{B})$ implies $w \in L(\mathcal{B})$. Hence, if all candidates belong to $L(\mathcal{B})$ then so do the discarded ones. The antichain heuristic allows a symbolic fixpoint computation of the remaining candidates [WDHR06].

Language inclusion can be decided by reasoning solely on ultimately periodic words (a.k.a. lasso words). So, the candidates are words of the form uv^ω, where $u \in \Sigma^*$ and $v \in \Sigma^+$ are called a stem and a period, respectively. [DGPR21] provides an algorithm that uses two quasiorders: one for the stems and one for the periods. Since using different quasiorders yields more pruning when searching for an inclusion violation, [DGM22b] considers using an unbounded number of quasiorders called FORQ: one for the stems and a family of quasiorders for the periods, each of them depending on a distinct stem. It is worth emphasizing that each quasiorder requires a fixpoint computation, and thus, the more quasiorders are handled, the more the antichain heuristic is leveraged.

The novelty of FORKLIFT lies in the use of FORQ to discard candidates. Below, we generalize FORQs for Büchi automata defined in [DGM22b] to support LimSup automata.

Definition 6.3.1. *Let $\mathcal{B} = (\Sigma, Q, \iota, \delta)$ be a* LimSup *automaton over the weights $W = \{\gamma(t) \mid t$ is a transition of $\mathcal{B}\}$. The structural FORQ of $\mathcal{B}$ is the pair $(\preceq^{\mathcal{B}}, \{\preceq^{\mathcal{B}}_u\}_{u \in \Sigma^*})$ where the quasiorders are defined by:*

$$u_1 \preceq^{\mathcal{B}} u_2 \iff \mathit{Tgt}_{\mathcal{B}}(u_1) \subseteq \mathit{Tgt}_{\mathcal{B}}(u_2)$$
$$v_1 \preceq^{\mathcal{B}}_u v_2 \iff \mathit{Cxt}_{\mathcal{B}}(\mathit{Tgt}_{\mathcal{B}}(u), v_1) \subseteq \mathit{Cxt}_{\mathcal{B}}(\mathit{Tgt}_{\mathcal{B}}(u), v_2)$$

with $\mathit{Tgt}_{\mathcal{B}} \colon \Sigma^* \to 2^Q$ and $\mathit{Cxt}_{\mathcal{B}} \colon 2^Q \times \Sigma^+ \to 2^{Q \times Q \times W}$ such that

$$\mathit{Tgt}_{\mathcal{B}}(u) = \{q' \in Q \mid \iota \xrightarrow{u}_{\mathcal{B}} q'\}$$
$$\mathit{Cxt}_{\mathcal{B}}(S, v) = \{(q, q', x) \mid q \in S, \rho = q \xrightarrow{v}_{\mathcal{B}} q', \text{ and } x \text{ is the maximum of the weight sequence of } \rho\}$$

The modification appears in the definition of $\mathrm{Cxt}_{\mathcal{B}}$ where x ranges over the weights of $\mathcal{B}$ instead of $\{\bot, \top\}$. Extending all the properties on structural FORQ established by [DGM22b] to this definition is straightforward, and implies the soundness of our inclusion algorithm for

```
Word* witEmpt, witSafe;
Automaton* A = new Automaton("A.txt");
Automaton* B = safetyClosure(A, LimInfAvg);
Automaton* C = livenessComponent(A, LimInfAvg);
bool flagEmpt = A->isNonEmpty(LimInfAvg, 5, &witEmpt);
bool flagSafe = A->isSafe(LimInfAvg, &witSafe);
```

Figure 6.4: An example usage of QuAK as a C++ library and its ability to compute witnesses for its results. The functions `isNonEmpty` and `isSafe` take an additional (optional) parameter for storing the stem and the period of the ultimately periodic word witnessing the algorithms' outputs.

LimSup automata. To use this algorithm for other classes of automata, we translate Inf, Sup, and LimInf automata to LimSup automata in PTIME for inclusion queries.

We highlight the remaining technical modifications below.

- Given a stem $u \in \Sigma^*$, the data structure used for the fixpoint computation of $\mathsf{Cxt}_{\mathcal{B}}$ carries (as in FORKLIFT) a period $v \in \Sigma^+$, a context $\mathsf{Cxt}_{\mathcal{B}}(\mathsf{Txt}_{\mathcal{B}}(u), v)$, and (in addition to FORKLIFT) the value of $\mathcal{A}$ over uv^ω.
- In FORKLIFT, the fixpoint computation of $\mathsf{Cxt}_{\mathcal{B}}$ does not leverage SCCs. A compilation option provides an implementation for QuAK that computes $\mathsf{Cxt}_{\mathcal{B}}$ while only considering intra-SCC transitions. We call this optimization `scc-search`.

6.3.4 Witness Computation

Internally, QuAK handles every problem either as inclusion checking or top-value computation. Therefore, to improve QuAK's informativeness and practical utility, we implemented these two algorithms with the capability of returning an ultimately periodic word witnessing the their results. Specifically, for inclusion checking (verifying that $\mathcal{A}(w) \leq \mathcal{B}(w)$ for all words w), the witness $\hat{w}$ satisfies $\mathcal{A}(\hat{w}) > \mathcal{B}(\hat{w})$, and for computing the top value $\top_{\mathcal{A}} = \sup_{w \in \Sigma^\omega} \mathcal{A}(w)$, the witness $\hat{w}$ meets $\mathcal{A}(\hat{w}) = \top_{\mathcal{A}}$.

Inclusion checking is implemented using an antichain-based algorithm as discussed above in Section 6.3.3. As this algorithm systematically searches for counterexamples, it inherently supports witness construction for negative instances. Top value computation is based on the standard graph-theoretic algorithms [CDH10b], with witness generation achieved via backtracking pointers.

Recall the nondeterministic limit-average automaton $\mathcal{A}$ and its safety-liveness decomposition from Figure 6.2. The first three lines of the code snippet in Figure 6.4 construct the automata $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$ as presented in Figure 6.2. The nonemptiness check returns `false`, and `witEmpt` points to an array storing $u = \mathit{hi}$ and $v = \mathit{hi}$ as $\top_{\mathcal{A}} = \mathcal{A}(\mathit{hi}^\omega) = 6$. Similarly, the safety check returns `false` and `witSafe` points to an array storing $u = v = \mathit{lo}$ as $\mathcal{B}(\mathit{lo}^\omega) = 4$ and $\mathcal{A}(\mathit{lo}^\omega) = 2$.

6.3.5 Constant-function Check for Limit-Average Automata

Checking whether a limit average automaton $\mathcal{A}$ is constant can be done by a reduction to the limitedness problem of distance automata [BHMS23, Thm. 3.7]. To simplify our implementation, we consider a reduction to the universality problem of LimInf automata. In

essence, our reduction follows [BHMS23, Thm. 3.7] except for in two points. First, after removing negative-weighted edges by Johnson's algorithm, instead of constructing a distance automaton, we flip the weights again to construct a limit average automaton (which resolves nondeterminism by $\sup$ and has the same value function as the input automaton). This yields an automaton $\mathcal{B}$ with non-positive transition weights. Then, we construct a LimInf automaton $\mathcal{C}$ by mapping negative weights of $\mathcal{B}$ to 0, and 0-valued weights to 1. Note that it may not hold that $\mathcal{B}(w) < 0$ iff $\mathcal{C}(w) < 1$ for all words w, but since $\mathcal{C}$ recognizes an ω-regular language, we can show that $\mathcal{B}$ is constant iff $\mathcal{C}$ is universal with respect to 1.

More specifically, the reduction goes as follows. Let $\mathcal{A}$ be a LimInfAvg (resp. LimSupAvg) automaton. First, construct $\mathcal{A}_1$ by subtracting $\top$ from all transition weights of $\mathcal{A}$. We have $\mathcal{A}_1(w) = \mathcal{A}(w) - \top$ for all words w. Then, construct $\mathcal{A}_2$ by multiplying by -1 all transition weights of $\mathcal{A}_1$, resolving nondeterminism by $\inf$, and taking the value function LimSupAvg (resp. LimInfAvg). We have $\mathcal{A}_2(w) = -\mathcal{A}_1(w)$ for all words w. Then, construct $\mathcal{A}_3$ by using Johnson's algorithm to remove transitions with negative weights of $\mathcal{A}_2$. We have $\mathcal{A}_2(w) > 0$ iff $\mathcal{A}_3(w) > 0$ for all words w. Then, construct $\mathcal{B}$ by multiplying by -1 all transition weights of $\mathcal{A}_3$, resolving nondeterminism by $\sup$, and taking the value function LimInfAvg (resp. LimSupAvg). We have $\mathcal{B}(w) = -\mathcal{A}_3(w)$ for all words w. Note that all transitions of $\mathcal{B}$ have non-positive weights. Finally, obtain $\mathcal{C}$ by updating the weights of $\mathcal{B}$ as follows and taking the value function LimInf: if a transition has weight 0, then its new weight is 1; otherwise (weight less than 0), then its new weight is 0.

By construction, $\mathcal{A}$ is constant $\top$ iff $\mathcal{B}$ is constant 0. We argue that $\mathcal{B}$ is constant iff $\mathcal{C}$ is universal (with respect to 1). If there is a word $w_1 \in \Sigma^\omega$ such that $\mathcal{B}(w_1) < 0$, then all runs of $\mathcal{B}$ over w_1 visit infinitely often some negative weight. Thus, $\mathcal{C}(w_1) < 1$ comes as a direct consequence of this implication. Note, however, that the reciprocal is not true, i.e., all runs of a word could visit infinitely often some negative weight while being mapped to 0 by $\mathcal{B}$. Now, if there is a word $w_2 \in \Sigma^\omega$ such that $\mathcal{C}(w_2) < 1$, then there exists also an ultimately periodic word $w \in \Sigma^\omega$ such that $\mathcal{C}(w) < 1$. This is because $\mathcal{C}$ is a co-Büchi automaton that defines a non-empty ω-regular language. Let w be of the form uv^ω. We define $x = |u|$, $y = |v|$, and let n be the number of states of $\mathcal{C}$. Suppose towards contradiction that some run of $\mathcal{C}$ over w visits only the weight 1 for $x + yn$ consecutive transitions. It implies that this run visits twice the same state at the end of the period v while visiting only the weight 1 in between, which exhibits another run of $\mathcal{C}$ over w of value 1, and thus leads to the contradiction $\mathcal{C}(w) = 1$. Hence, all runs of $\mathcal{C}$ over w periodically visit the weight 0 after $x + yn$ transitions. Since $\mathcal{B}$ differs from $\mathcal{C}$ only in transition weights, all runs of $\mathcal{B}$ over w periodically visit some negative weight after $x + yn$ transitions, therefore $\mathcal{B}(w) < 0$. In conclusion, $\mathcal{A}$ defines a constant function iff $\mathcal{C}$ is universal (with respect to 1). Note that we can directly construct $\mathcal{C}$ from $\mathcal{A}$ in PTime.

6.3.6 Monitoring

Given a specification represented as a deterministic quantitative automaton $\mathcal{A}$, QuAK is able create a monitor object that stores an array of top values (storing the top value of $\mathcal{A}^q$ for each state q of $\mathcal{A}$), an array of bottom values (storing the bottom value of $\mathcal{A}^q$ for each state q of $\mathcal{A}$), and a pointer to the current state of $\mathcal{A}$ (initialized as the initial state of $\mathcal{A}$). A monitor object can read input letters incrementally while getting the next state q of $\mathcal{A}$ and maintaining the lowest and highest values achievable from q, namely, the bottom and top values of $\mathcal{A}^q$. In addition, we implement running average monitors for limit average automata.

6.4 Experimental Evaluation

We evaluated QuAK in a set of experiments. In particular, we measure the performance our antichain-based inclusion algorithm and compare it to the standard algorithm based on repeated reduction to language inclusion of Büchi automata. These experiments include also the measurement of the impact of the `scc-search` optimization described in Section 6.3. Next, we evaluate the runtime of checking if an automaton defines a constant funcion. Finally, we use QuAK to runtime monitor the smoothness of a controller for a drone to show that the tool can be used in the context of quantitative runtime monitoring.

Setup QuAK was compiled with -O3 and link-time optimizations enabled. The `scc-search` optimization was enabled for all experiments except a part of those that aimed at evaluating this optimization (Section 6.4.1). All experiments ran on machines with *AMD EPYC* CPU with the frequency 3.1 GHz. The time limit was set to 100 s wall time.

Benchmarks Because of the lack of benchmarks for quantitative automata, we used randomly generated quantitative automata. All automata are complete (i.e., every state has an outgoing transition for each symbol in the alphabet) and have weights between -10 and 10 chosen uniformly at random. An automaton that has n states can have up to $n|\Sigma| + 2n + 1$ edges where Σ is the alphabet. As a result, the generated automata are nondeterministic. The number of states and the size of alphabet differ in the experiments and are always explicitly mentioned.

6.4.1 Comparing Inclusion Algorithms

In this subsection, we compare the standard approach to compute the quantitative automata inclusion (referred to as *Standard*) with our antichain-based inclusion algorithm (referred to as *Antichains*). The implementation of the standard approach uses the boolean version of FORKLIFT to decide the inclusion of boolean automata. Both algorithms are implemented in QuAK.

Figure 6.5 shows the CPU time of running *Standard* and *Antichains* algorithms for Val $\in$ $\{\text{Sup}, \text{LimSup}\}$. We used 100 random automata with 2–32 states and with 2-symbol alphabet. Algorithms were ran for each possible pair of the automata, which results in 10000 inclusion checks. In the plots, we show only the runs where at least one algorithm decided the inclusion.

The algorithm *Antichains* is almost always faster, often significantly, and it can finish in a lot of cases when *Standard* reaches the time limit (points on the blue dashed line). The *Standard* algorithm internally runs (the boolean version of) *Antichains* algorithm multiple times for each weight, and therefore it is expected that *Antichains* should be faster most of the times.

Evaluating Optimizations of Inclusion Algorithms The results in Figure 6.5 are for QuAK that is compiled with the `scc-search` optimization (see Section 6.3.3). Plots in Figure 6.6 show that this optimization significantly improves the runtime. The plot on the left shows how many instances (the x axis) can be decided given the time limit is set to the value on the y axis. The optimization allows to decide nearly 2000 more instances in under 2 seconds. The plot on the right shows that the optimization also hurts in some cases. Nevertheless, it helps with approximately 90% of the considered automata.

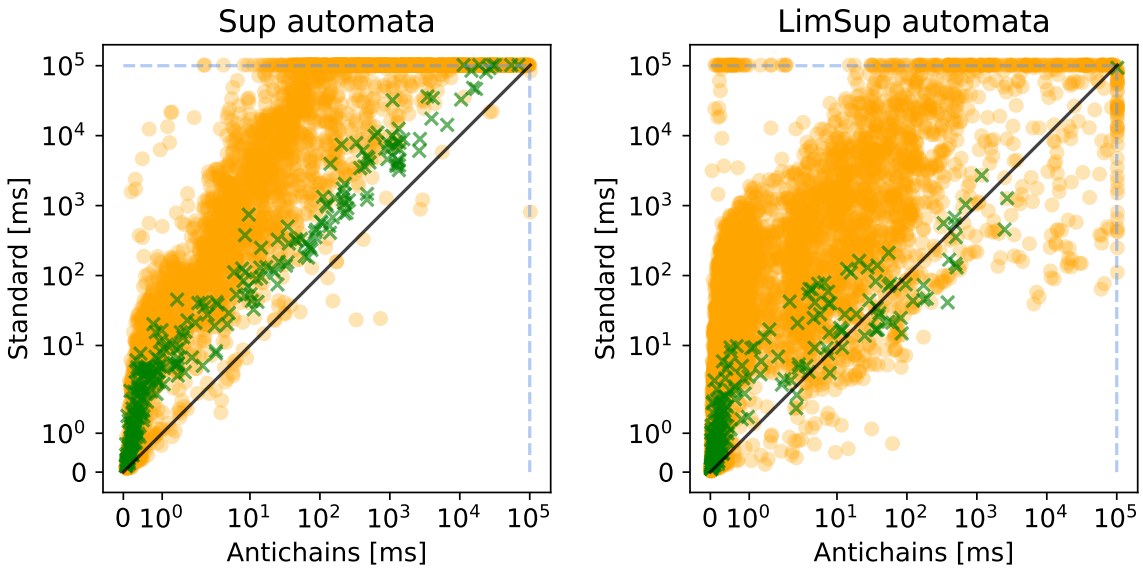


Figure 6.5: CPU time in milliseconds of running *Antichains* (x axis) and *Standard* (y axis) inclusion algorithms on random automata with 2–32 states where at least one algorithm finished within time limit. The alphabet has 2. Orange dots are for pairs of automata that are not included and green crosses are for included automata. The scatter plot on the left is for Sup and on the right for LimSup value function.

6.4.2 Evaluating Constant-function Checking

To evaluate the constant-function checking algorithm for limit-average automata, we generated 1000 random automata with a 4-symbol alphabet and 1–100 states. The results of running the algorithm on these automata are summarized in Figure 6.7.

The computational complexity of the algorithm increases steeply: for larger automata, the result is typically computed either quickly or not at all. While the algorithm times out on many instances, the results suggest that deciding whether an automaton is constant remains feasible in certain cases. Notably, all instances that do not time out in our experiments fall into one of two categories: they are either deterministic, for which the problem is in PTime, or the resulting co-Büchi automaton has a very low density of accepting edges, for which the antichain-based algorithm finds witnesses for non-universality more easily.

6.4.3 Runtime Monitoring

We experimented with using quantitative automata for runtime monitoring. Our use case is to monitor the smoothness of controllers of cyber-physical systems (CPS) [MMMS21], which means that the actions issued by a CPS controller should always cause only a relatively small change in the state of the CPS. For example, a controller of a drone should not instruct it to immediately change to the opposite of the current direction. Controllers that are not smooth can lead to increased energy consumption or even hardware failures [MMMS21].

We monitored a flying drone in a simulated environment. For simplicity, we assumed that the drone is a point with mass 1 and its energy consumption is equal to the sum of forces generated by the thrust of its engines. Each action issued by a controller is a pair of integers (x, y) that represents the acceleration vector (on a 2D plane), with $-10 \leq x, y \leq 10$. Therefore, the alphabet Σ has 441 symbols. The monitor computes the running average of weights of the

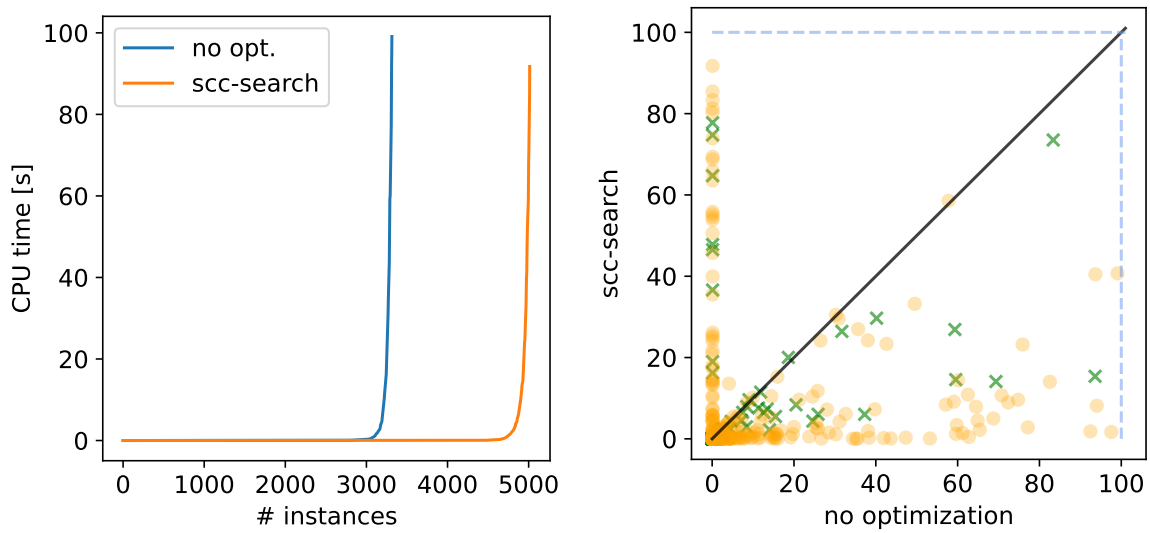


Figure 6.6: CPU time of running *Antichains* algorithm with and without the `scc-search` optimization on the benchmarks from Figure 6.5. In the left plot, the x axis shows how many instances the inclusion algorithms are able to decide given the time limit on the y axis. The right plot compares the runtime per instance. There, orange dots are for pairs of automata that are not included and green crosses are for included automata. The plots are for Sup automata and time is in seconds.

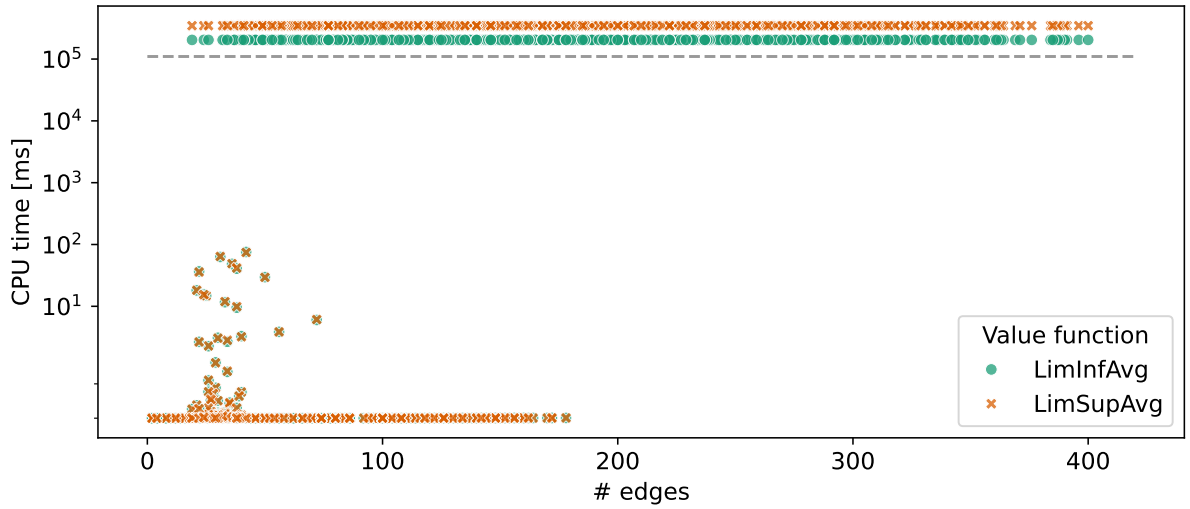
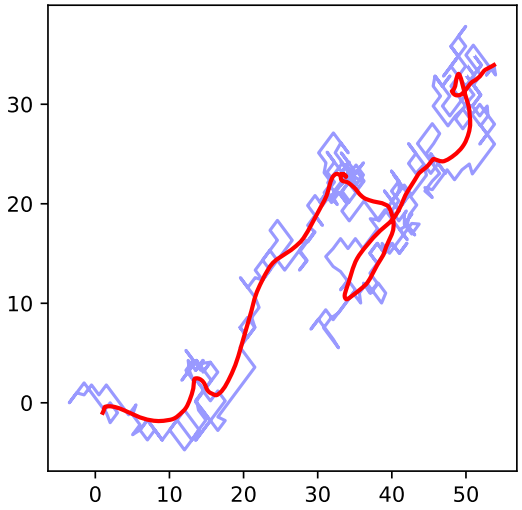


Figure 6.7: CPU time of deciding if an automaton defines a constant function. Points representing timeouts were moved above the timeout line (the dashed line) and separated (with no particular order with respect to the vertical axis).

automaton $\mathcal{A}_M$ that has one state for each symbol from Σ, and from each state q there is an outgoing edge $q \xrightarrow{q':x} q'$ to any other state q' under the symbol q'. In other words, the states remember the last issued action. The weight x of each transition going from q to q' is the distance between q and q'. In total, the automaton $\mathcal{A}_M$ has 441 states and 194481 edges.

The initial mission of the drone was to get from the point $(0, 0)$ to $(1000, 1000)$ (with no obstacles) using a controller that every 0.1 s issues a command to accelerate toward the target. However, a random deviation taken from the normal distribution with mean 0 and standard deviation either 1 or 5 (this is a parameter) is applied to both acceleration coordinates at



	Original		Smoothed	
Controller	*EC*	*Score*	*EC*	*Score*
sd=1	9.95	12.83	3.40	0.15
sd=5	12.42	15.97	2.42	0.82

Figure 6.8: Results of monitoring the smoothness of a drone controller on an erratic and smoothed trajectory (resp., *Original* and *Smoothed*). The situation is depicted on the left where the erratic trajectory is blue and the smoothed one is red. Only a part of the trajectories is shown. In the table on the right, *Score* is the value computed by the monitor and *EC* is the energy consumption of the drone on the trajectory. The lower is the score, the smoother should be the trajectory. All numbers are averages from 3 simulations.

every step. The magnitude of the acceleration is also random, skewed toward the maximum acceleration value 10. If the resulting acceleration along a coordinate is greater (lower) than 10 (resp., -10), it is set to 10 (resp., -10).

The rather chaotic controller described above models an imperfect controller and results in navigating the drone along an erratic trajectory. We ran another mission where the drone followed the previously taken erratic trace that has been smoothed using gradient ascent. The situation is depicted on the left in Figure 6.8, and the results of monitoring the trajectories is on the right in the same figure. The monitor correctly assigns lower scores to smoother trajectories, which directly corresponds to the difference in energy consumption (EC).

6.5 Conclusion

We presented QuAK, our software tool for automating quantitative automata analysis QuAK supports Inf, Sup, LimInf, LimSup, LimInfAvg, and LimSupAvg automata, along with algorithms not only for standard decision problems but also for safety and liveness, by reducing these problems to checking automaton inclusion or computing top values. Future work aims to improve the tool's scalability and applicability while exploring more efficient verification methods. One promising avenue is the development of symbolic approaches to efficiently manage large state spaces. Another key direction involves extending the tool to support

additional formalisms, such as various types of discounted-sum automata [Bok24], mean-payoff automaton expressions [CDE+10], and nested quantitative automata [CHO17]. In parallel with these efforts, developing novel verification methods specifically tailored to the safety fragments of expressive quantitative formalisms presents an exciting research direction.

CHAPTER 7

Conclusions and Future Work

In this thesis, we focused on the classification and monitoring of quantitative properties. We considered resource-precision tradeoffs of quantitative and approximate monitors central and accordingly presented a theory of monitorability that allows the formal analysis of quantitative properties and monitors from this perspective. In particular, we made the following contributions.

- In Chapter 3, we defined different forms of quantitative and approximate limit monitorability and showed how these definitions naturally extend existing boolean monitorability notions. From these definitions, we developed a framework for evaluating monitors by analyzing trade-offs between their precision and resource use. We illustrated, through examples, that providing monitors with more resources, like additional registers or states, can result in improved approximations.

  In Chapter 4, we proposed an abstract view of quantitative monitors as an equivalence class on finite words and a function that maps each class to a value, allowing to assess monitor precision and resource use. Based on this abstraction, we showed that optimal approximate monitors, in terms of resource use, are not unique and they may not greedily minimize the size of their equivalence relation. We also illustrated how our approach enables a formal analysis of resource-precision tradeoffs for quantitative monitors. For instance, we presented properties that admit infinite hierarchies of approximate monitors and others where reducing resources inevitably worsens the limit error.

- In Chapter 5, we formalized quantitative generalizations of safety, liveness, co-safety, and co-liveness, and established that every quantitative property can be decomposed as the pointwise minimum of a safety and a liveness property (and similarly, as the pointwise maximum of a co-safety and a co-liveness property). We identified the connection between quantitative safety and topological continuity and provided characterizations that explain how quantitative safety and liveness relate back to their boolean counterparts. We proposed the notions of approximate safety and co-safety and proved that every quantitative property that is approximately safe and co-safe admits a finite-state approximate monitor. Then, focusing on quantitative properties definable by finite-state quantitative automata, we showed a close relationship between verifying if a quantitative automaton defines a constant function and deciding its safety and liveness; we presented algorithms to decide this property for standard classes of quantitative automata. Finally,

we provided algorithms for computing the safety closure of quantitative automata, deciding their safety and liveness, and decomposing them into their safety and liveness components.

- In Chapter 6, we presented the first software tool QuAK for automating the analysis and monitoring of quantitative automata. In addition to implementing the standard quantitative automata algorithms from the literature, such as nonemptiness, universality, and inclusion checking, we integrated most of our algorithms from Chapter 5 to allow deciding safety and liveness and to decompose quantitative automata accordingly. We also showed how QuAK can be used for monitoring and evaluated it empirically to demonstrate its practical effectiveness.

We conclude the thesis with a discussion of future research directions.

An absolute notion of resource use The theory of quantitative and approximate monitorability we introduced in Chapter 3 provided a relative approach to reasoning about a monitor's quality: the key notion of precision focused only on whether a monitor gets closer to the property value compared to another monitor. We addressed this in Chapter 4 by considering properties that are defined as limits of values of finite words—the so-called limit properties—and defining a monitor's precision as the worst-case distance of its verdict values to these property values. Therefore, we are able to speak of a monitor's quality in absolute terms, not only relative to another monitor.

A current drawback of our monitorability framework is the lack of means to reason about a monitor's resource use in absolute terms beyond the number of states (of a finite-state monitor) or registers (of a register monitor). This measure might suffice for finite-state monitors, but it is not fine-grained enough for infinite-state ones. Although our notion of resource optimality offers a partial solution, it still lacks the nuance for reasoning about the resource use of non-optimal monitors. Therefore, there is still a need to develop the resource-use side of our theory further. The exploration may benefit from the plethora of work on streaming algorithms, where such resource use (and precision) concerns are analyzed in great detail.

Synthesizing approximate monitors In this thesis, we mainly demonstrated the "descriptive" side of our framework: We identified classes of properties that are amenable to resource-precision tradeoffs in approximate monitoring, we clarified various difficulties in developing and analyzing approximate monitors, we showed the existence of arbitrarily precise finite-state monitors for properties that are both safe and co-safe, and so on. These contributions are valuable for understanding the approximate monitoring setting and recognizing the opportunities and challenges for instantiating our theoretical framework in practice, which we have so far focused little on.

An important part we aim to explore further is the "constructive" side of our theory, focusing on the synthesis of approximate monitors under resource and precision constraints. We believe this is a crucial first step to bridge the gap between theory and practice of approximate monitoring. One may start with a simple setting where fully automatic methods can be developed, e.g., for finite-state properties and monitors, and push it further by considering semi-automatic or incomplete methods.

Making quantitative safety (and liveness) practical In Chapter 5, we introduced the notions of safety and liveness for quantitative properties and studied the two ends of the

spectrum: first, without making any assumptions on the specification language, and then considering only the classes of quantitative properties defined by finite-state quantitative automata. In the boolean setting, safety fragments of expressive formalisms can be verified or monitored efficiently, while liveness fragments require heavier methods. Our results also indicate a similar potential, for example, every limit-average automaton that defines a safety property can be expressed as an Inf automaton and thus enjoys many closure and decidability properties.

We plan to investigate classes of quantitative properties beyond finite-state, study their safety and liveness, and provide efficient verification and monitoring algorithms for them. A first step in this direction may focus on nested quantitative automata [CHO17], which are able to express many interesting properties such as response time. Moreover, another direction may explore approaches to verify and monitor subclasses of finite-state liveness properties. These efforts may be complemented by developing QuAK from Chapter 6 by implementing the resulting algorithms.

Quantitative hyperproperties So far, we have only considered trace properties, which are described as functions of individual system executions: every execution is given a value independently of other executions. Such properties have traditionally been the focus of verification efforts, but they lack the ability to specify properties that consider how multiple executions relate. Hyperproperties address this gap—they describe system properties as functions of sets of executions: each possible implementation is given a value that reflects how its executions relate [CS10]. In the quantitative setting, while a trace property can specify the average response time of a server's execution, a hyperproperty can specify the worst-case average response time of its implementations.

We plan to define and systematically study such quantitative hyperproperties, including exploring their safety and liveness. While a general view, as in the first part of Chapter 5, can help, developing and investigating formalisms for specifying quantitative hyperproperties will be crucial.

Bibliography

[AAC+05] Chris Allan, Pavel Avgustinov, Aske Simon Christensen, Laurie Hendren, Sascha Kuzins, Ondřej Lhoták, Oege De Moor, Damien Sereni, Ganesh Sittampalam, and Julian Tibble. Adding trace matching with free variables to aspectj. *ACM SIGPLAN Notices*, 40(10):345–364, 2005.

[AAF+19a] Luca Aceto, Antonis Achilleos, Adrian Francalanza, Anna Ingólfsdóttir, and Karoliina Lehtinen. Adventures in monitorability: from branching to linear time and back again. *Proc. ACM Program. Lang.*, 3(POPL):52:1–52:29, 2019.

[AAF+19b] Luca Aceto, Antonis Achilleos, Adrian Francalanza, Anna Ingólfsdóttir, and Karoliina Lehtinen. An operational guide to monitorability. In Peter Csaba Ölveczky and Gwen Salaün, editors, *Software Engineering and Formal Methods*, pages 433–453, Cham, 2019. Springer International Publishing.

[AAF+21a] Luca Aceto, Antonis Achilleos, Adrian Francalanza, Anna Ingólfsdóttir, and Karoliina Lehtinen. The best a monitor can do. In Christel Baier and Jean Goubault-Larrecq, editors, *29th EACSL Annual Conference on Computer Science Logic, CSL 2021, January 25-28, 2021, Ljubljana, Slovenia (Virtual Conference)*, volume 183 of *LIPIcs*, pages 7:1–7:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[AAF+21b] Luca Aceto, Antonis Achilleos, Adrian Francalanza, Anna Ingólfsdóttir, and Karoliina Lehtinen. An operational guide to monitorability with applications to regular properties. *Softw. Syst. Model.*, 20(2):335–361, 2021.

[ABK14] Shaull Almagor, Udi Boker, and Orna Kupferman. Discounting in LTL. In Erika Ábrahám and Klaus Havelund, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 20th International Conference, TACAS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014. Proceedings*, volume 8413 of *Lecture Notes in Computer Science*, pages 424–439. Springer, 2014.

[ABK22] Shaull Almagor, Udi Boker, and Orna Kupferman. What's decidable about weighted automata? *Inf. Comput.*, 282:104651, 2022.

[AC10] Rajeev Alur and Pavol Cerný. Expressiveness of streaming string transducers. In Kamal Lodaya and Meena Mahajan, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2010, December 15-18, 2010, Chennai, India*, volume 8 of *LIPIcs*, pages 1–12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2010.

[AC11] Rajeev Alur and Pavol Cerný. Streaming transducers for algorithmic verification of single-pass list-processing programs. In Thomas Ball and Mooly Sagiv, editors, *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA, January 26-28, 2011*, pages 599–610. ACM, 2011.

[ACD+21] Giorgio Audrito, Roberto Casadei, Ferruccio Damiani, Volker Stolz, and Mirko Viroli. Adaptive distributed monitors of spatial properties for cyber-physical systems. *J. Syst. Softw.*, 175:110908, 2021.

[AD94] Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.

[AD17] Rajeev Alur and Loris D'Antoni. Streaming tree transducers. *J. ACM*, 64(5):31:1–31:55, 2017.

[ADD+13] Rajeev Alur, Loris D'Antoni, Jyotirmoy V. Deshmukh, Mukund Raghothaman, and Yifei Yuan. Regular functions and cost register automata. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013*, pages 13–22. IEEE Computer Society, 2013.

[ADL14] Natasha Alechina, Mehdi Dastani, and Brian Logan. Norm approximation for imperfect monitors. In Ana L. C. Bazzan, Michael N. Huhns, Alessio Lomuscio, and Paul Scerri, editors, *International conference on Autonomous Agents and Multi-Agent Systems, AAMAS '14, Paris, France, May 5-9, 2014*, pages 117–124. IFAAMAS/ACM, 2014.

[ADX01] Paul Ammann, Wei Ding, and Daling Xu. Using a model checker to test safety properties. In *Proceedings Seventh IEEE International Conference on Engineering of Complex Computer Systems*, pages 212–221. IEEE, 2001.

[AFM+20] Rajeev Alur, Dana Fisman, Konstantinos Mamouras, Mukund Raghothaman, and Caleb Stanford. Streamable regular transductions. *Theor. Comput. Sci.*, 807:15–41, 2020.

[AFR16] Rajeev Alur, Dana Fisman, and Mukund Raghothaman. Regular programming for quantitative properties of data streams. In Peter Thiemann, editor, *Programming Languages and Systems - 25th European Symposium on Programming, ESOP 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, volume 9632 of *Lecture Notes in Computer Science*, pages 15–40. Springer, 2016.

[AFT12] Rajeev Alur, Emmanuel Filiot, and Ashutosh Trivedi. Regular transformations of infinite strings. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25-28, 2012*, pages 65–74. IEEE Computer Society, 2012.

[AH93] Rajeev Alur and Thomas A. Henzinger. Real-time logics: Complexity and expressiveness. *Inf. Comput.*, 104(1):35–77, 1993.

[AHK02] Rajeev Alur, Thomas A Henzinger, and Orna Kupferman. Alternating-time temporal logic. *Journal of the ACM (JACM)*, 49(5):672–713, 2002.

[AKL13] Benjamin Aminof, Orna Kupferman, and Robby Lampert. Rigorous approximated determinization of weighted automata. *Theor. Comput. Sci.*, 480:104–117, 2013.

[Alb03] Susanne Albers. Online algorithms: a survey. *Mathematical Programming*, 97(1):3–26, 2003.

[ALFS11] Yashwanth Annpureddy, Che Liu, Georgios Fainekos, and Sriram Sankaranarayanan. S-taliro: A tool for temporal logic falsification for hybrid systems. In Parosh Aziz Abdulla and K. Rustan M. Leino, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 17th International Conference, TACAS 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings*, volume 6605 of *Lecture Notes in Computer Science*, pages 254–257. Springer, 2011.

[AMP95] Eugene Asarin, Oded Maler, and Amir Pnueli. Reachability analysis of dynamical systems having piecewise-constant derivatives. *Theor. Comput. Sci.*, 138(1):35–65, 1995.

[AMS17] Rajeev Alur, Konstantinos Mamouras, and Caleb Stanford. Automata-based stream processing. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 112:1–112:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[AMS19] Rajeev Alur, Konstantinos Mamouras, and Caleb Stanford. Modular quantitative monitoring. *Proc. ACM Program. Lang.*, 3(POPL):50:1–50:31, January 2019.

[AO16] Paul Ammann and Jeff Offutt. *Introduction to software testing*. Cambridge University Press, 2016.

[APM19] Houssam Abbas, Yash Vardhan Pant, and Rahul Mangharam. Temporal logic robustness for general signal classes. In Necmiye Ozay and Pavithra Prabhakar, editors, *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2019, Montreal, QC, Canada, April 16-18, 2019*, pages 45–56. ACM, 2019.

[AS85] Bowen Alpern and Fred B. Schneider. Defining liveness. *Inf. Process. Lett.*, 21(4):181–185, 1985.

[AS87] Bowen Alpern and Fred B. Schneider. Recognizing safety and liveness. *Distributed Comput.*, 2(3):117–126, 1987.

[BBB+22] Haniel Barbosa, Clark Barrett, Martin Brain, Gereon Kremer, Hanna Lachnitt, Makai Mann, Abdalrhman Mohamed, Mudathir Mohamed, Aina Niemetz, Andres Nötzli, et al. cvc5: A versatile and industrial-strength smt solver. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 415–442. Springer, 2022.

[BBC+06] Thomas Ball, Ella Bounimova, Byron Cook, Vladimir Levin, Jakob Lichtenberg, Con McGarvey, Bohus Ondrusek, Sriram K Rajamani, and Abdullah Ustuner.

Thorough static analysis of device drivers. *ACM SIGOPS Operating Systems Review*, 40(4):73–85, 2006.

[BCFK15] Tomáš Brázdil, Krishnendu Chatterjee, Vojtěch Forejt, and Antonín Kučera. Multigain: A controller synthesis tool for mdps with multiple mean-payoff objectives. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 181–187. Springer, 2015.

[BCHJ09] Roderick Bloem, Krishnendu Chatterjee, Thomas A. Henzinger, and Barbara Jobstmann. Better quality in synthesis through quantitative objectives. In Ahmed Bouajjani and Oded Maler, editors, *Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France, June 26 - July 2, 2009. Proceedings*, volume 5643 of *Lecture Notes in Computer Science*, pages 140–156. Springer, 2009.

[BCHK14] Udi Boker, Krishnendu Chatterjee, Thomas A. Henzinger, and Orna Kupferman. Temporal specifications with accumulative values. *ACM Trans. Comput. Log.*, 15(4):27:1–27:25, 2014.

[BCJ18] Roderick Bloem, Krishnendu Chatterjee, and Barbara Jobstmann. *Graph Games and Reactive Synthesis*, pages 921–962. Springer, 2018.

[BDH+20] David Basin, Thibault Dardinier, Lukas Heimes, Srđan Krstić, Martin Raszyk, Joshua Schneider, and Dmitriy Traytel. A formally verified, optimized monitor for metric first-order dynamic logic. In *Automated Reasoning: 10th International Joint Conference, IJCAR 2020, Paris, France, July 1–4, 2020, Proceedings, Part I 10*, pages 432–453. Springer, 2020.

[Ber07] Antonia Bertolino. Software testing research: Achievements, challenges, dreams. In *Future of Software Engineering (FOSE'07)*, pages 85–103. IEEE, 2007.

[BFFR18] Ezio Bartocci, Yliès Falcone, Adrian Francalanza, and Giles Reger. Introduction to runtime verification. In Ezio Bartocci and Yliès Falcone, editors, *Lectures on Runtime Verification - Introductory and Advanced Topics*, volume 10457 of *Lecture Notes in Computer Science*, pages 1–33. Springer, 2018.

[BFH+12] Howard Barringer, Yliès Falcone, Klaus Havelund, Giles Reger, and David E. Rydeheard. Quantified event automata: Towards expressive and efficient runtime monitors. In Dimitra Giannakopoulou and Dominique Méry, editors, *FM 2012: Formal Methods - 18th International Symposium, Paris, France, August 27-31, 2012. Proceedings*, volume 7436 of *Lecture Notes in Computer Science*, pages 68–84. Springer, 2012.

[BFMU17] Alexey Bakhirkin, Thomas Ferrère, Oded Maler, and Dogan Ulus. On the quantitative semantics of regular expressions over real-valued signals. In Alessandro Abate and Gilles Geeraerts, editors, *Formal Modeling and Analysis of Timed Systems - 15th International Conference, FORMATS 2017, Berlin, Germany, September 5-7, 2017, Proceedings*, volume 10419 of *Lecture Notes in Computer Science*, pages 189–206. Springer, 2017.

[BFS+20] Jan Baumeister, Bernd Finkbeiner, Sebastian Schirmer, Maximilian Schwenger, and Christoph Torens. Rtlola cleared for take-off: Monitoring autonomous aircraft.

In Shuvendu K. Lahiri and Chao Wang, editors, *Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part II*, volume 12225 of *Lecture Notes in Computer Science*, pages 28–39. Springer, 2020.

[BG99] Glenn Bruns and Patrice Godefroid. Model checking partial state spaces with 3-valued temporal logics. In Nicolas Halbwachs and Doron A. Peled, editors, *Computer Aided Verification, 11th International Conference, CAV '99, Trento, Italy, July 6-10, 1999, Proceedings*, volume 1633 of *Lecture Notes in Computer Science*, pages 274–287. Springer, 1999.

[BH12] Udi Boker and Thomas A. Henzinger. Approximate determinization of quantitative automata. In Deepak D'Souza, Telikepalli Kavitha, and Jaikumar Radhakrishnan, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2012, December 15-17, 2012, Hyderabad, India*, volume 18 of *LIPIcs*, pages 362–373. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012.

[BH14] Udi Boker and Thomas A. Henzinger. Exact and approximate determinization of discounted-sum automata. *Log. Methods Comput. Sci.*, 10(1), 2014.

[BH21] Udi Boker and Guy Hefetz. Discounted-sum automata with multiple discount factors. In Christel Baier and Jean Goubault-Larrecq, editors, *29th EACSL Annual Conference on Computer Science Logic, CSL 2021, January 25-28, 2021, Ljubljana, Slovenia (Virtual Conference)*, volume 183 of *LIPIcs*, pages 12:1–12:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[BH23] Udi Boker and Guy Hefetz. On the comparison of discounted-sum automata with multiple discount factors. In Orna Kupferman and Pawel Sobocinski, editors, *Foundations of Software Science and Computation Structures - 26th International Conference, FoSSaCS 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2023, Paris, France, April 22-27, 2023, Proceedings*, volume 13992 of *Lecture Notes in Computer Science*, pages 371–391. Springer, 2023.

[BHHK03] Christel Baier, Boudewijn Haverkort, Holger Hermanns, and J-P Katoen. Model-checking algorithms for continuous-time markov chains. *IEEE Transactions on software engineering*, 29(6):524–541, 2003.

[BHKZ12] David Basin, Matúš Harvan, Felix Klaedtke, and Eugen Zălinescu. Monpoly: Monitoring usage-control policies. In *Runtime Verification: Second International Conference, RV 2011, San Francisco, CA, USA, September 27-30, 2011, Revised Selected Papers 2*, pages 360–364. Springer, 2012.

[BHMS23] Udi Boker, Thomas A. Henzinger, Nicolas Mazzocchi, and N. Ege Saraç. Safety and liveness of quantitative automata. In Guillermo A. Pérez and Jean-François Raskin, editors, *34th International Conference on Concurrency Theory, CONCUR 2023, September 18-23, 2023, Antwerp, Belgium*, volume 279 of *LIPIcs*, pages 17:1–17:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

[BHMS25] Udi Boker, Thomas A. Henzinger, Nicolas Mazzocchi, and N. Ege Saraç. Safety and liveness of quantitative properties and automata. *Logical Methods in Computer Science*, Volume 21, Issue 2, Apr 2025.

[BHO15] Udi Boker, Thomas A. Henzinger, and Jan 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. IEEE Computer Society, 2015.

[BJA+21] James Bornholt, Rajeev Joshi, Vytautas Astrauskas, Brendan Cully, Bernhard Kragl, Seth Markle, Kyle Sauri, Drew Schleit, Grant Slatton, Serdar Tasiran, et al. Using lightweight formal methods to validate a key-value storage node in amazon s3. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, pages 836–850, 2021.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT press, 2008.

[BKMZ15] David A. Basin, Felix Klaedtke, Samuel Müller, and Eugen Zalinescu. Monitoring metric first-order temporal properties. *J. ACM*, 62(2):15:1–15:45, 2015.

[BKZ17] David A Basin, Felix Klaedtke, and Eugen Zalinescu. The monpoly monitoring tool. *RV-CuBES*, 3:19–28, 2017.

[BLS06] Andreas Bauer, Martin Leucker, and Christian Schallhart. Monitoring of real-time properties. In S. Arun-Kumar and Naveen Garg, editors, *FSTTCS 2006: Foundations of Software Technology and Theoretical Computer Science, 26th International Conference, Kolkata, India, December 13-15, 2006, Proceedings*, volume 4337 of *Lecture Notes in Computer Science*, pages 260–272. Springer, 2006.

[BLS07] Andreas Bauer, Martin Leucker, and Christian Schallhart. The good, the bad, and the ugly, but how ugly is ugly? In Oleg Sokolsky and Serdar Tasiran, editors, *Runtime Verification, 7th International Workshop, RV 2007, Vancouver, Canada, March 13, 2007, Revised Selected Papers*, volume 4839 of *Lecture Notes in Computer Science*, pages 126–138. Springer, 2007.

[BLS10] Andreas Bauer, Martin Leucker, and Christian Schallhart. Comparing LTL semantics for runtime verification. *J. Log. Comput.*, 20(3):651–674, 2010.

[BLS11] Andreas Bauer, Martin Leucker, and Christian Schallhart. Runtime verification for LTL and TLTL. *ACM Trans. Softw. Eng. Methodol.*, 20(4):14:1–14:64, 2011.

[BMM14] Patricia Bouyer, Nicolas Markey, and Raj Mohan Matteplackel. Averaging in LTL. In Paolo Baldan and Daniele Gorla, editors, *CONCUR 2014 - Concurrency Theory - 25th International Conference, CONCUR 2014, Rome, Italy, September 2-5, 2014. Proceedings*, volume 8704 of *Lecture Notes in Computer Science*, pages 266–280. Springer, 2014.

[BMNS24] Borzoo Bonakdarpour, Anik Momtaz, Dejan Nickovic, and N. Ege Saraç. Approximate distributed monitoring under partial synchrony: Balancing speed & accuracy. In Erika Ábrahám and Houssam Abbas, editors, *Runtime Verification - 24th International Conference, RV 2024, Istanbul, Turkey, October 15-17, 2024, Proceedings*, volume 15191 of *Lecture Notes in Computer Science*, pages 282–301. Springer, 2024.

[BMR+18] Patricia Bouyer, Nicolas Markey, Mickael Randour, Kim G. Larsen, and Simon Laursen. Average-energy games. *Acta Informatica*, 55(2):91–127, 2018.

[BNF13] Borzoo Bonakdarpour, Samaneh Navabpour, and Sebastian Fischmeister. Time-triggered runtime verification. *Formal Methods Syst. Des.*, 43(1):29–60, 2013.

[Bok21] Udi Boker. Quantitative vs. weighted automata. In Paul C. Bell, Patrick Totzke, and Igor Potapov, editors, *Reachability Problems - 15th International Conference, RP 2021, Liverpool, UK, October 25-27, 2021, Proceedings*, volume 13035 of *Lecture Notes in Computer Science*, pages 3–18. Springer, 2021.

[Bok24] Udi Boker. Discounted-sum automata with real-valued discount factors. In Pawel Sobocinski, Ugo Dal Lago, and Javier Esparza, editors, *Proceedings of the 39th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2024, Tallinn, Estonia, July 8-11, 2024*, pages 15:1–15:14. ACM, 2024.

[BRH08] Howard Barringer, David Rydeheard, and Klaus Havelund. Rule systems for run-time monitoring: from eagle to ruler. *Journal of Logic and Computation*, 20(3):675–706, 2008.

[BS23] Jason P. Bell and Daniel Smertnig. Computing the linear hull: Deciding deterministic? and unambiguous? for weighted automata over fields. In *38th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2023, Boston, MA, USA, June 26-29, 2023*, pages 1–13. IEEE, 2023.

[BV19] Suguman Bansal and Moshe Y. Vardi. Safety and co-safety comparator automata for discounted-sum inclusion. In Isil Dillig and Serdar Tasiran, editors, *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I*, volume 11561 of *Lecture Notes in Computer Science*, pages 60–78. Springer, 2019.

[CB02] Paul Caspi and Albert Benveniste. Toward an approximation theory for computerised control. In Alberto L. Sangiovanni-Vincentelli and Joseph Sifakis, editors, *Embedded Software, Second International Conference, EMSOFT 2002, Grenoble, France, October 7-9, 2002, Proceedings*, volume 2491 of *Lecture Notes in Computer Science*, pages 294–304. Springer, 2002.

[CC77] Patrick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In Robert M. Graham, Michael A. Harrison, and Ravi Sethi, editors, *Conference Record of the Fourth ACM Symposium on Principles of Programming Languages, Los Angeles, California, USA, January 1977*, pages 238–252. ACM, 1977.

[CCG+02] Alessandro Cimatti, Edmund Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella. Nusmv 2: An opensource tool for symbolic model checking. In *Computer Aided Verification: 14th International Conference, CAV 2002 Copenhagen, Denmark, July 27–31, 2002 Proceedings 14*, pages 359–364. Springer, 2002.

[CD11] Krishnendu Chatterjee and Laurent Doyen. Energy and mean-payoff parity markov decision processes. In Filip Murlak and Piotr Sankowski, editors, *Mathematical Foundations of Computer Science 2011 - 36th International Symposium, MFCS*

2011, Warsaw, Poland, August 22-26, 2011. Proceedings, volume 6907 of *Lecture Notes in Computer Science*, pages 206–218. Springer, 2011.

[CDD+15] Cristiano Calcagno, Dino Distefano, Jérémy Dubreil, Dominik Gabi, Pieter Hooimeijer, Martino Luca, Peter O'Hearn, Irene Papakonstantinou, Jim Purbrick, and Dulma Rodriguez. Moving fast with software verification. In *NASA Formal Methods Symposium*, pages 3–11. Springer, 2015.

[CDE+10] Krishnendu Chatterjee, Laurent Doyen, Herbert Edelsbrunner, Thomas A. Henzinger, and Philippe Rannou. Mean-payoff automaton expressions. In Paul Gastin and François Laroussinie, editors, *CONCUR 2010 - Concurrency Theory, 21th International Conference, CONCUR 2010, Paris, France, August 31-September 3, 2010. Proceedings*, volume 6269 of *Lecture Notes in Computer Science*, pages 269–283. Springer, 2010.

[CDH09] Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Alternating weighted automata. In Miroslaw Kutylowski, Witold Charatonik, and Maciej Gebala, editors, *Fundamentals of Computation Theory, 17th International Symposium, FCT 2009, Wroclaw, Poland, September 2-4, 2009. Proceedings*, volume 5699 of *Lecture Notes in Computer Science*, pages 3–13. Springer, 2009.

[CDH10a] Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Expressiveness and closure properties for quantitative languages. *Log. Methods Comput. Sci.*, 6(3), 2010.

[CDH10b] Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Quantitative languages. *ACM Trans. Comput. Log.*, 11(4):23:1–23:38, July 2010.

[CGD02] Marsha Chechik, Arie Gurfinkel, and Benet Devereux. chi-chek: A multi-valued model-checker. In Ed Brinksma and Kim Guldstrand Larsen, editors, *Computer Aided Verification, 14th International Conference, CAV 2002,Copenhagen, Denmark, July 27-31, 2002, Proceedings*, volume 2404 of *Lecture Notes in Computer Science*, pages 505–509. Springer, 2002.

[CGJ+00] Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In *Computer Aided Verification: 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000. Proceedings 12*, pages 154–169. Springer, 2000.

[CGJP13] Radu Calinescu, Simos Gerasimou, Kenneth Johnson, and Colin Paterson. Using runtime quantitative verification to provide assurance evidence for self-adaptive software - advances, applications and research challenges. In Rogério de Lemos, David Garlan, Carlo Ghezzi, and Holger Giese, editors, *Software Engineering for Self-Adaptive Systems III. Assurances - International Seminar, Dagstuhl Castle, Germany, December 15-19, 2013, Revised Selected and Invited Papers*, volume 9640 of *Lecture Notes in Computer Science*, pages 223–248. Springer, 2013.

[CGKM12] Radu Calinescu, Carlo Ghezzi, Marta Z. Kwiatkowska, and Raffaela Mirandola. Self-adaptive software needs quantitative verification at runtime. *Commun. ACM*, 55(9):69–77, 2012.

[CHL+18] Lukas Convent, Sebastian Hungerecker, Martin Leucker, Torben Scheffel, Malte Schmitz, and Daniel Thoma. Tessla: Temporal stream-based specification language. In Tiago Massoni and Mohammad Reza Mousavi, editors, *Formal Methods: Foundations and Applications - 21st Brazilian Symposium, SBMF 2018, Salvador, Brazil, November 26-30, 2018, Proceedings*, volume 11254 of *Lecture Notes in Computer Science*, pages 144–162. Springer, 2018.

[CHMS24] Marek Chalupa, Thomas A. Henzinger, Nicolas Mazzocchi, and N. Ege Saraç. Quak: Quantitative automata kit. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation. Software Engineering Methodologies - 12th International Symposium, ISoLA 2024, Crete, Greece, October 27-31, 2024, Proceedings, Part IV*, volume 15222 of *Lecture Notes in Computer Science*, pages 3–20. Springer, 2024.

[CHMS25] Marek Chalupa, Thomas A. Henzinger, Nicolas Mazzocchi, and N. Ege Saraç. Automating the analysis of quantitative automata with quak, 2025.

[CHO16] Krishnendu Chatterjee, Thomas A. Henzinger, and Jan Otop. Quantitative monitor automata. In Xavier Rival, editor, *Static Analysis - 23rd International Symposium, SAS 2016, Edinburgh, UK, September 8-10, 2016, Proceedings*, volume 9837 of *Lecture Notes in Computer Science*, pages 23–38. Springer, 2016.

[CHO17] Krishnendu Chatterjee, Thomas A. Henzinger, and Jan Otop. Nested weighted automata. *ACM Trans. Comput. Log.*, 18(4):31:1–31:44, 2017.

[CHO19] Krishnendu Chatterjee, Thomas A. Henzinger, and Jan Otop. Quantitative automata under probabilistic semantics. *Log. Methods Comput. Sci.*, 15(3), 2019.

[CHR12] Pavol Cerný, Thomas A. Henzinger, and Arjun Radhakrishna. Simulation distances. *Theor. Comput. Sci.*, 413(1):21–35, 2012.

[CHV+18] Edmund M Clarke, Thomas A Henzinger, Helmut Veith, Roderick Bloem, et al. *Handbook of model checking*, volume 10. Springer, 2018.

[CKL04] Edmund Clarke, Daniel Kroening, and Flavio Lerda. A tool for checking ansi-c programs. In *Tools and Algorithms for the Construction and Analysis of Systems: 10th International Conference, TACAS 2004, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004, Barcelona, Spain, March 29-April 2, 2004. Proceedings 10*, pages 168–176. Springer, 2004.

[Cla97] Edmund M. Clarke. Model cheking. In S. Ramesh and G. Sivakumar, editors, *Foundations of Software Technology and Theoretical Computer Science, 17th Conference, Kharagpur, India, December 18-20, 1997, Proceedings*, volume 1346 of *Lecture Notes in Computer Science*, pages 54–56. Springer, 1997.

[CLKB04] Jeffrey Considine, Feifei Li, George Kollios, and John W. Byers. Approximate aggregation techniques for sensor databases. In Z. Meral Özsoyoglu and Stanley B. Zdonik, editors, *Proceedings of the 20th International Conference on Data Engineering, ICDE 2004, 30 March - 2 April 2004, Boston, MA, USA*, pages 449–460. IEEE Computer Society, 2004.

[CM17] Mmanu Chaturvedi and Ross M. McConnell. A note on finding minimum mean cycle. *Inf. Process. Lett.*, 127:21–22, 2017.

[CMP93] Edward Chang, Zohar Manna, and Amir Pnueli. The safety-progress classification. In Friedrich L. Bauer, Wilfried Brauer, and Helmut Schwichtenberg, editors, *Logic and Algebra of Specification*, pages 143–202, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

[Cou96] Patrick Cousot. Abstract interpretation. *ACM Comput. Surv.*, 28(2):324–328, 1996.

[CR09] Feng Chen and Grigore Rosu. Parametric trace slicing and monitoring. In Stefan Kowalewski and Anna Philippou, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 15th International Conference, TACAS 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings*, volume 5505 of *Lecture Notes in Computer Science*, pages 246–261. Springer, 2009.

[CS10] Michael R. Clarkson and Fred B. Schneider. Hyperproperties. *J. Comput. Secur.*, 18(6):1157–1210, 2010.

[CTT21] Alessandro Cimatti, Chun Tian, and Stefano Tonetta. Assumption-based runtime verification of infinite-state systems. In Lu Feng and Dana Fisman, editors, *Runtime Verification - 21st International Conference, RV 2021, Virtual Event, October 11-14, 2021, Proceedings*, volume 12974 of *Lecture Notes in Computer Science*, pages 207–227. Springer, 2021.

[dAHM03] Luca de Alfaro, Thomas A. Henzinger, and Rupak Majumdar. Discounting the future in systems theory. In Jos C. M. Baeten, Jan Karel Lenstra, Joachim Parrow, and Gerhard J. Woeginger, editors, *Automata, Languages and Programming, 30th International Colloquium, ICALP 2003, Eindhoven, The Netherlands, June 30 - July 4, 2003. Proceedings*, volume 2719 of *Lecture Notes in Computer Science*, pages 1022–1037. Springer, 2003.

[DDG+10] Aldric Degorre, Laurent Doyen, Raffaella Gentilini, Jean-François Raskin, and Szymon Torunczyk. Energy and mean-payoff games with imperfect information. In Anuj Dawar and Helmut Veith, editors, *Computer Science Logic, 24th International Workshop, CSL 2010, 19th Annual Conference of the EACSL, Brno, Czech Republic, August 23-27, 2010. Proceedings*, volume 6247 of *Lecture Notes in Computer Science*, pages 260–274. Springer, 2010.

[DDLS13] Akim Demaille, Alexandre Duret-Lutz, Sylvain Lombardy, and Jacques Sakarovitch. Implementation concepts in vaucanson 2. In Stavros Konstantinidis, editor, *Implementation and Application of Automata - 18th International Conference, CIAA 2013, Halifax, NS, Canada, July 16-19, 2013. Proceedings*, volume 7982 of *Lecture Notes in Computer Science*, pages 122–133. Springer, 2013.

[DDS17] Ankush Desai, Tommaso Dreossi, and Sanjit A Seshia. Combining model checking and runtime verification for safe robotics. In *International Conference on Runtime Verification*, pages 172–189. Springer, 2017.

[DFM13] Alexandre Donzé, Thomas Ferrere, and Oded Maler. Efficient robust monitoring for stl. In *International Conference on Computer Aided Verification*, pages 264–279. Springer, 2013.

[DGM22a] Kyveli Doveri, Pierre Ganty, and Nicolas Mazzocchi. FORKLIFT (v1.0). Zenodo, 2022. maintained at https://github.com/Mazzocchi/FORKLIFT.

[DGM22b] Kyveli Doveri, Pierre Ganty, and Nicolas Mazzocchi. Forq-based language inclusion formal testing. In Sharon Shoham and Yakir Vizel, editors, *Computer Aided Verification - 34th International Conference, CAV 2022, Haifa, Israel, August 7-10, 2022, Proceedings, Part II*, volume 13372 of *Lecture Notes in Computer Science*, pages 109–129. Springer, 2022.

[DGPR21] Kyveli Doveri, Pierre Ganty, Francesco Parolini, and Francesco Ranzato. Inclusion testing of büchi automata based on well-quasiorders. In Serge Haddad and Daniele Varacca, editors, *32nd International Conference on Concurrency Theory, CONCUR 2021, August 24-27, 2021, Virtual Conference*, volume 203 of *LIPIcs*, pages 3:1–3:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[Dij76] Edsger W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976.

[Din01] M Dindoš. Generalized cantor sets and sets of sums of convergent alternating series. *Journal of Applied Analysis*, 7(1):131–150, 2001.

[DJK+99] Siddhartha R Dalal, Ashish Jain, Nachimuthu Karunanithi, JM Leaton, Christopher M Lott, Gardner C Patton, and Bruce M Horowitz. Model-based testing in practice. In *Proceedings of the 21st international conference on Software engineering*, pages 285–294, 1999.

[DJKV17] Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen, and Matthias Volk. A storm is coming: A modern probabilistic model checker. In Rupak Majumdar and Viktor Kuncak, editors, *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II*, volume 10427 of *Lecture Notes in Computer Science*, pages 592–600. Springer, 2017.

[DK21] Manfred Droste and Dietrich Kuske. Weighted automata. In Jean-Éric Pin, editor, *Handbook of Automata Theory*, pages 113–150. European Mathematical Society Publishing House, Zürich, Switzerland, 2021.

[DKV09] Manfred Droste, Werner Kuich, and Heiko Vogler. *Handbook of weighted automata*. Springer Science & Business Media, 2009.

[DKW08] Vijay D'silva, Daniel Kroening, and Georg Weissenbacher. A survey of automated techniques for formal software verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(7):1165–1178, 2008.

[DL09] Stéphane Demri and Ranko Lazic. LTL with the freeze quantifier and register automata. *ACM Trans. Comput. Log.*, 10(3):16:1–16:30, 2009.

[DL14] Volker Diekert and Martin Leucker. Topology, monitorable properties and runtime verification. *Theor. Comput. Sci.*, 537:29–41, 2014.

[DLT13a] Normann Decker, Martin Leucker, and Daniel Thoma. Impartiality and anticipation for monitoring of visibly context-free properties. In Axel Legay and Saddek Bensalem, editors, *Runtime Verification - 4th International Conference, RV 2013, Rennes, France, September 24-27, 2013. Proceedings*, volume 8174 of *Lecture Notes in Computer Science*, pages 183–200. Springer, 2013.

[DLT13b] Normann Decker, Martin Leucker, and Daniel Thoma. junit rv–adding runtime verification to junit. In *NASA Formal Methods: 5th International Symposium, NFM 2013, Moffett Field, CA, USA, May 14-16, 2013. Proceedings 5*, pages 459–464. Springer, 2013.

[DLT15] Xiaoning Du, Yang Liu, and Alwen Tiu. Trace-length independent runtime monitoring of quantitative policies in LTL. In Nikolaj S. Bjørner and Frank S. de Boer, editors, *FM 2015: Formal Methods - 20th International Symposium, Oslo, Norway, June 24-26, 2015, Proceedings*, volume 9109 of *Lecture Notes in Computer Science*, pages 231–247. Springer, 2015.

[DLT16] Normann Decker, Martin Leucker, and Daniel Thoma. Monitoring modulo theories. *International Journal on Software Tools for Technology Transfer*, 18(2):205–225, 2016.

[DM12] Manfred Droste and Ingmar Meinecke. Weighted automata and weighted MSO logics for average and long-time behaviors. *Inf. Comput.*, 220:44–59, 2012.

[DMB08] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

[DMKA+15] Leonardo De Moura, Soonho Kong, Jeremy Avigad, Floris Van Doorn, and Jakob von Raumer. The lean theorem prover (system description). In *Automated Deduction-CADE-25: 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings 25*, pages 378–388. Springer, 2015.

[Don10] Alexandre Donzé. Breach, A toolbox for verification and parameter synthesis of hybrid systems. In Tayssir Touili, Byron Cook, and Paul B. Jackson, editors, *Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings*, volume 6174 of *Lecture Notes in Computer Science*, pages 167–170. Springer, 2010.

[dSS+05] Ben d'Angelo, Sriram Sankaranarayanan, César Sánchez, Will Robinson, Bernd Finkbeiner, Henny B Sipma, Sandeep Mehrotra, and Zohar Manna. Lola: runtime monitoring of synchronous systems. In *12th International Symposium on Temporal Representation and Reasoning (TIME'05)*, pages 166–174. IEEE, 2005.

[DSS16] Loris D'Antoni, Roopsha Samanta, and Rishabh Singh. Qlose: Program repair with quantitative objectives. In Swarat Chaudhuri and Azadeh Farzan, editors, *Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part II*, volume 9780 of *Lecture Notes in Computer Science*, pages 383–401. Springer, 2016.

[EFH+03] Cindy Eisner, Dana Fisman, John Havlicek, Yoad Lustig, Anthony McIsaac, and David Van Campenhout. Reasoning with temporal logic on truncated paths. In Warren A. Hunt Jr. and Fabio Somenzi, editors, *Computer Aided Verification, 15th International Conference, CAV 2003, Boulder, CO, USA, July 8-12, 2003, Proceedings*, volume 2725 of *Lecture Notes in Computer Science*, pages 27–39. Springer, 2003.

[EN98] E Allen Emerson and Kedar S Namjoshi. On model checking for non-deterministic infinite-state systems. In *Proceedings. Thirteenth Annual IEEE Symposium on Logic in Computer Science (Cat. No. 98CB36226)*, pages 70–80. IEEE, 1998.

[FAA+17] Adrian Francalanza, Luca Aceto, Antonis Achilleos, Duncan Paul Attard, Ian Cassar, Dario Della Monica, and Anna Ingólfsdóttir. A foundation for runtime monitoring. In Shuvendu K. Lahiri and Giles Reger, editors, *Runtime Verification - 17th International Conference, RV 2017, Seattle, WA, USA, September 13-16, 2017, Proceedings*, volume 10548 of *Lecture Notes in Computer Science*, pages 8–29. Springer, 2017.

[FFM12] Yliès Falcone, Jean-Claude Fernandez, and Laurent Mounier. What can you verify and enforce at runtime? *Int. J. Softw. Tools Technol. Transf.*, 14(3):349–382, 2012.

[FFS+19] Peter Faymonville, Bernd Finkbeiner, Malte Schledjewski, Maximilian Schwenger, Marvin Stenger, Leander Tentrup, and Hazem Torfah. Streamlab: stream-based monitoring of cyber-physical systems. In *International Conference on Computer Aided Verification*, pages 421–431. Springer, 2019.

[FFST16] Peter Faymonville, Bernd Finkbeiner, Sebastian Schirmer, and Hazem Torfah. A stream-based specification language for network monitoring. In Yliès Falcone and César Sánchez, editors, *Runtime Verification - 16th International Conference, RV 2016, Madrid, Spain, September 23-30, 2016, Proceedings*, volume 10012 of *Lecture Notes in Computer Science*, pages 152–168. Springer, 2016.

[FHK20] Thomas Ferrère, Thomas A. Henzinger, and Bernhard Kragl. Monitoring event frequencies. In Maribel Fernández and Anca Muscholl, editors, *28th EACSL Annual Conference on Computer Science Logic, CSL 2020, January 13-16, 2020, Barcelona, Spain*, volume 152 of *LIPIcs*, pages 20:1–20:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[FHS18] Thomas Ferrère, Thomas A. Henzinger, and N. Ege Saraç. A theory of register monitors. In Anuj Dawar and Erich Grädel, editors, *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 394–403. ACM, 2018.

[FK18] Rachel Faran and Orna Kupferman. Spanning the spectrum from safety to liveness. *Acta Informatica*, 55(8):703–732, 2018.

[FKN+11] Vojtěch Forejt, Marta Kwiatkowska, Gethin Norman, David Parker, and Hongyang Qu. Quantitative multi-objective verification for probabilistic systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 112–127. Springer, 2011.

[FL14] Uli Fahrenberg and Axel Legay. The quantitative linear-time-branching-time spectrum. *Theor. Comput. Sci.*, 538:54–69, 2014.

[Flo67] RW Floyd. Assigning meanings to programs, 19. *American Mathematical Society, Providence, RI*, 1967.

[FLS08] Marco Faella, Axel Legay, and Mariëlle Stoelinga. Model checking quantitative linear time logic. In Alessandro Aldini and Christel Baier, editors, *Proceedings of the Sixth Workshop on Quantitative Aspects of Programming Languages, QAPL 2008, Budapest, Hungary, March 29-30, 2008*, volume 220 of *Electronic Notes in Theoretical Computer Science*, pages 61–77. Elsevier, 2008.

[FMFR11] Yliès Falcone, Laurent Mounier, Jean-Claude Fernandez, and Jean-Luc Richier. Runtime enforcement monitors: composition, synthesis, and enforcement abilities. *Formal Methods Syst. Des.*, 38(3):223–262, 2011.

[FP06] Georgios E. Fainekos and George J. Pappas. Robustness of temporal logic specifications. In Klaus Havelund, Manuel Núñez, Grigore Rosu, and Burkhart Wolff, editors, *Formal Approaches to Software Testing and Runtime Verification, First Combined International Workshops, FATES 2006 and RV 2006, Seattle, WA, USA, August 15-16, 2006, Revised Selected Papers*, volume 4262 of *Lecture Notes in Computer Science*, pages 178–192. Springer, 2006.

[FP09] Georgios E. Fainekos and George J. Pappas. Robustness of temporal logic specifications for continuous-time signals. *Theor. Comput. Sci.*, 410(42):4262–4291, 2009.

[FP18] Nathan Fulton and André Platzer. Safe reinforcement learning via formal methods: Toward safe control through proof and learning. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 6485–6492. AAAI Press, 2018.

[Fra21] Adrian Francalanza. A theory of monitors. *Inf. Comput.*, 281:104704, 2021.

[FS04] Bernd Finkbeiner and Henny Sipma. Checking finite traces using alternating automata. *Formal Methods in System Design*, 24:101–127, 2004.

[GDPT13] Radu Grigore, Dino Distefano, Rasmus Lerchedahl Petersen, and Nikos Tzevelekos. Runtime verification based on register automata. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 260–276. Springer, 2013.

[GG99] Theodore W Gamelin and Robert Everist Greene. *Introduction to topology*. Courier Corporation, 1999.

[GH01] Dimitra Giannakopoulou and Klaus Havelund. Automata-based verification of temporal properties on running programs. In *16th IEEE International Conference on Automated Software Engineering (ASE 2001), 26-29 November 2001, Coronado Island, San Diego, CA, USA*, pages 412–416. IEEE Computer Society, 2001.

[GHK+03] Gerhard Gierz, Karl Heinrich Hofmann, Klaus Keimel, Jimmie D Lawson, Michael Mislove, and Dana S Scott. *Continuous lattices and domains*, volume 93. Cambridge university press, 2003.

[GLM+08] Patrice Godefroid, Michael Y Levin, David A Molnar, et al. Automated whitebox fuzz testing. In *NDSS*, volume 8, pages 151–166, 2008.

[GMDC+18] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. Ai2: Safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE symposium on security and privacy (SP)*, pages 3–18. IEEE, 2018.

[GMUW08] Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom. *Database Systems: The Complete Book*. Prentice Hall Press, USA, 2 edition, 2008.

[GPVW95] Rob Gerth, Doron A. Peled, Moshe Y. Vardi, and Pierre Wolper. Simple on-the-fly automatic verification of linear temporal logic. In Piotr Dembinski and Marek Sredniawa, editors, *Protocol Specification, Testing and Verification XV, Proceedings of the Fifteenth IFIP WG6.1 International Symposium on Protocol Specification, Testing and Verification, Warsaw, Poland, June 1995*, volume 38 of *IFIP Conference Proceedings*, pages 3–18. Chapman & Hall, 1995.

[GS21] Felipe Gorostiaga and César Sánchez. Nested monitors: monitors as expressions to build monitors. In *International Conference on Runtime Verification*, pages 164–183. Springer, 2021.

[GS22] Felipe Gorostiaga and César Sánchez. Monitorability of expressive verdicts. In Jyotirmoy V. Deshmukh, Klaus Havelund, and Ivan Perez, editors, *NASA Formal Methods - 14th International Symposium, NFM 2022, Pasadena, CA, USA, May 24-27, 2022, Proceedings*, volume 13260 of *Lecture Notes in Computer Science*, pages 693–712. Springer, 2022.

[GS24] Felipe Gorostiaga and César Sánchez. General monitorability of totally ordered verdict domains. *Innovations in Systems and Software Engineering*, pages 1–14, 2024.

[GV13] Giuseppe De Giacomo and Moshe Y. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In Francesca Rossi, editor, *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, pages 854–860. IJCAI/AAAI, 2013.

[Has82] K. Hashiguchi. Limitedness theorem on finite automata with distance functions. *Journal of computer and system sciences*, 24(2):233–244, 1982.

[Has00] K. Hashiguchi. New upper bounds to the limitedness of distance automata. *Theoretical Computer Science*, 233(1-2):19–32, 2000.

[Hav15] Klaus Havelund. Rule-based runtime verification revisited. *Int. J. Softw. Tools Technol. Transf.*, 17(2):143–170, 2015.

[Hen96] Thomas A. Henzinger. The theory of hybrid automata. In *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science, New Brunswick, New Jersey, USA, July 27-30, 1996*, pages 278–292. IEEE Computer Society, 1996.

[Hen13] Thomas A. Henzinger. Quantitative reactive modeling and verification. *Comput. Sci. Res. Dev.*, 28(4):331–344, November 2013.

[HH94] Thomas A. Henzinger and Pei-Hsin Ho. HYTECH: the cornell hybrid technology tool. In Panos J. Antsaklis, Wolf Kohn, Anil Nerode, and Shankar Sastry, editors, *Hybrid Systems II, Proceedings of the Third International Workshop on Hybrid Systems, Ithaca, NY, USA, October 1994*, volume 999 of *Lecture Notes in Computer Science*, pages 265–293. Springer, 1994.

[HK12] Shulamit Halamish and Orna Kupferman. Approximating deterministic lattice automata. In Supratik Chakraborty and Madhavan Mukund, editors, *Automated Technology for Verification and Analysis - 10th International Symposium, ATVA 2012, Thiruvananthapuram, India, October 3-6, 2012. Proceedings*, volume 7561 of *Lecture Notes in Computer Science*, pages 27–41. Springer, 2012.

[HKMS23] Thomas A. Henzinger, Pavol Kebis, Nicolas Mazzocchi, and N. Ege Saraç. Regular methods for operator precedence languages. In Kousha Etessami, Uriel Feige, and Gabriele Puppis, editors, *50th International Colloquium on Automata, Languages, and Programming, ICALP 2023, July 10-14, 2023, Paderborn, Germany*, volume 261 of *LIPIcs*, pages 129:1–129:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

[HKMS25] Thomas A. Henzinger, Pavol Kebis, Nicolas Mazzocchi, and N. Ege Saraç. Quantitative language automata. In Patricia Bouyer and Jaco van de Pol, editors, *36th International Conference on Concurrency Theory, CONCUR 2025, August 26-29, 2025, Aarhus, Denmark*, volume 348 of *LIPIcs*, pages 21:1–21:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025.

[HLS20] Thomas A. Henzinger, Anna Lukina, and Christian Schilling. Outside the box: Abstraction-based monitoring of neural networks. In Giuseppe De Giacomo, Alejandro Catalá, Bistra Dilkina, Michela Milano, Senén Barro, Alberto Bugarín, and Jérôme Lang, editors, *ECAI 2020 - 24th European Conference on Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain, August 29 - September 8, 2020 - Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020)*, volume 325 of *Frontiers in Artificial Intelligence and Applications*, pages 2433–2440. IOS Press, 2020.

[HMS22] Thomas A. Henzinger, Nicolas Mazzocchi, and N. Ege Saraç. Abstract monitors for quantitative specifications. In Thao Dang and Volker Stolz, editors, *Runtime Verification - 22nd International Conference, RV 2022, Tbilisi, Georgia, September 28-30, 2022, Proceedings*, volume 13498 of *Lecture Notes in Computer Science*, pages 200–220. Springer, 2022.

[HMS23] Thomas A. Henzinger, Nicolas Mazzocchi, and N. Ege Saraç. Quantitative safety and liveness. In Orna Kupferman and Pawel Sobocinski, editors, *Foundations of Software Science and Computation Structures - 26th International Conference, FoSSaCS 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2023, Paris, France, April 22-27, 2023, Proceedings*, volume 13992 of *Lecture Notes in Computer Science*, pages 349–370. Springer, 2023.

[HMS24] Thomas A. Henzinger, Nicolas Mazzocchi, and N. Ege Saraç. Strategic dominance: A new preorder for nondeterministic processes. In Rupak Majumdar and Alexandra Silva, editors, *35th International Conference on Concurrency Theory, CONCUR 2024, September 9-13, 2024, Calgary, Canada*, volume 311 of *LIPIcs*, pages 29:1–29:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.

[HO13] Thomas A. Henzinger and Jan Otop. From model checking to model measuring. In Pedro R. D'Argenio and Hernán C. Melgratti, editors, *CONCUR 2013 - Concurrency Theory - 24th International Conference, CONCUR 2013, Buenos Aires, Argentina, August 27-30, 2013. Proceedings*, volume 8052 of *Lecture Notes in Computer Science*, pages 273–287. Springer, 2013.

[Hoa69] C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12(10):576–580, 1969.

[HOW14] Hsi-Ming Ho, Joël Ouaknine, and James Worrell. Online monitoring of metric temporal logic. In Borzoo Bonakdarpour and Scott A. Smolka, editors, *Runtime Verification - 5th International Conference, RV 2014, Toronto, ON, Canada, September 22-25, 2014. Proceedings*, volume 8734 of *Lecture Notes in Computer Science*, pages 178–192. Springer, 2014.

[HP18] Klaus Havelund and Doron Peled. Runtime verification: From propositional to first-order temporal logic. In Christian Colombo and Martin Leucker, editors, *Runtime Verification - 18th International Conference, RV 2018, Limassol, Cyprus, November 10-13, 2018, Proceedings*, volume 11237 of *Lecture Notes in Computer Science*, pages 90–112. Springer, 2018.

[HP22] Klaus Havelund and Doron Peled. On monitoring linear temporal properties. *Formal Methods Syst. Des.*, 60(3):405–425, 2022.

[HP23] Klaus Havelund and Doron Peled. Monitorability for runtime verification. In Panagiotis Katsaros and Laura Nenzi, editors, *Runtime Verification - 23rd International Conference, RV 2023, Thessaloniki, Greece, October 3-6, 2023, Proceedings*, volume 14245 of *Lecture Notes in Computer Science*, pages 447–460. Springer, 2023.

[HPPR18] Paul Hunter, Arno Pauly, Guillermo A. Pérez, and Jean-François Raskin. Mean-payoff games with partial observation. *Theor. Comput. Sci.*, 735:82–110, 2018.

[HPU20] Klaus Havelund, Doron Peled, and Dogan Ulus. First-order temporal logic monitoring with bdds. *Formal Methods in System Design*, 56(1):1–21, 2020.

[HR86] Hendrik Jan Hoogeboom and Grzegorz Rozenberg. Infinitary languages: Basic theory an applications to concurrent systems. In J. W. de Bakker, Willem P. de Roever, and Grzegorz Rozenberg, editors, *Current Trends in Concurrency, Overviews and Tutorials*, volume 224 of *Lecture Notes in Computer Science*, pages 266–342. Springer, 1986.

[HR01] Klaus Havelund and Grigore Rosu. Monitoring programs using rewriting. In *16th IEEE International Conference on Automated Software Engineering (ASE 2001), 26-29 November 2001, Coronado Island, San Diego, CA, USA*, pages 135–143. IEEE Computer Society, 2001.

[HR02] Klaus Havelund and Grigore Rosu. Synthesizing monitors for safety properties. In Joost-Pieter Katoen and Perdita Stevens, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 8th International Conference, TACAS 2002, Held as Part of the Joint European Conference on Theory and Practice of Software, ETAPS 2002, Grenoble, France, April 8-12, 2002, Proceedings*, volume 2280 of *Lecture Notes in Computer Science*, pages 342–356. Springer, 2002.

[HR04a] Klaus Havelund and Grigore Roșu. Efficient monitoring of safety properties. *International Journal on Software Tools for Technology Transfer*, 6(2):158–173, 2004.

[HR04b] Klaus Havelund and Grigore Roșu. An overview of the runtime verification tool java pathexplorer. *Formal methods in system design*, 24:189–215, 2004.

[HRTZ18] Klaus Havelund, Giles Reger, Daniel Thoma, and Eugen Zalinescu. Monitoring events that carry data. In Ezio Bartocci and Yliès Falcone, editors, *Lectures on Runtime Verification - Introductory and Advanced Topics*, volume 10457 of *Lecture Notes in Computer Science*, pages 61–102. Springer, 2018.

[HS00] Gerard J. Holzmann and Margaret H. Smith. Automating software feature verification. *Bell Labs Tech. J.*, 5(2):72–87, 2000.

[HS20] Thomas A. Henzinger and N. Ege Saraç. Monitorability under assumptions. In Jyotirmoy Deshmukh and Dejan Nickovic, editors, *Runtime Verification - 20th International Conference, RV 2020, Los Angeles, CA, USA, October 6-9, 2020, Proceedings*, volume 12399 of *Lecture Notes in Computer Science*, pages 3–18. Springer, 2020.

[HS21] Thomas A. Henzinger and N. Ege Saraç. Quantitative and approximate monitoring. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*, pages 1–14. IEEE, 2021. © 2021 IEEE. Reprinted, with permission, from Thomas A. Henzinger and N. Ege Saraç, "Quantitative and Approximate Monitoring," 2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), June 2021.

[HT87] Thilo Hafer and Wolfgang Thomas. Computation tree logic ctl* and path quantifiers in the monadic theory of the binary tree. In Thomas Ottmann, editor, *Automata, Languages and Programming, 14th International Colloquium, ICALP87, Karlsruhe, Germany, July 13-17, 1987, Proceedings*, volume 267 of *Lecture Notes in Computer Science*, pages 269–279. Springer, 1987.

[JBG+18] Stefan Jakšić, Ezio Bartocci, Radu Grosu, Thang Nguyen, and Dejan Ničković. Quantitative monitoring of stl with edit distance. *Formal methods in system design*, 53(1):83–112, 2018.

[JGS93] Neil D. Jones, Carsten K. Gomard, and Peter Sestoft. *Partial evaluation and automatic program generation*. Prentice Hall international series in computer science. Prentice Hall, 1993.

[JMLR12] Dongyun Jin, Patrick O'Neil Meredith, Choonghwan Lee, and Grigore Roșu. Javamop: Efficient parametric runtime monitoring framework. In *2012 34th International Conference on Software Engineering (ICSE)*, pages 1427–1430. IEEE, 2012.

[Joh77] D. B. Johnson. Efficient algorithms for shortest paths in sparse networks. *Journal of the ACM (JACM)*, 24(1):1–13, 1977.

[KAB+17] Bettina Könighofer, Mohammed Alshiekh, Roderick Bloem, Laura R. Humphrey, Robert Könighofer, Ufuk Topcu, and Chao Wang. Shield synthesis. *Formal Methods Syst. Des.*, 51(2):332–361, 2017.

[Kar78] Richard M. Karp. A characterization of the minimum cycle mean in a digraph. *Discret. Math.*, 23(3):309–311, 1978.

[KBD+17] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *Computer Aided Verification: 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I 30*, pages 97–117. Springer, 2017.

[KHF20] Sean Kauffman, Klaus Havelund, and Sebastian Fischmeister. What can we monitor over unreliable channels? *International Journal on Software Tools for Technology Transfer*, pages 1–24, 2020.

[KKL+02] Moonjoo Kim, Sampath Kannan, Insup Lee, Oleg Sokolsky, and Mahesh Viswanathan. Computational analysis of run-time monitoring: Fundamentals of java-mac1 1this research was supported in part by onr n00014-97-1-0505, nsf ccr-9988409, nsf ccr-0086147, nsf cise-9703220, and aro daad19-01-1-0473. *Electronic Notes in Theoretical Computer Science*, 70(4):80–94, 2002. RV'02, Runtime Verification 2002 (FLoC Satellite Event).

[KL07] Orna Kupferman and Yoad Lustig. Lattice automata. In Byron Cook and Andreas Podelski, editors, *Verification, Model Checking, and Abstract Interpretation, 8th International Conference, VMCAI 2007, Nice, France, January 14-16, 2007, Proceedings*, volume 4349 of *Lecture Notes in Computer Science*, pages 199–213. Springer, 2007.

[KMB18] Ayrat Khalimov, Benedikt Maderbacher, and Roderick Bloem. Bounded synthesis of register transducers. In Shuvendu K. Lahiri and Chao Wang, editors, *Automated Technology for Verification and Analysis - 16th International Symposium, ATVA 2018, Los Angeles, CA, USA, October 7-10, 2018, Proceedings*, volume 11138 of *Lecture Notes in Computer Science*, pages 494–510. Springer, 2018.

[KNP02] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. PRISM: probabilistic symbolic model checker. In Tony Field, Peter G. Harrison, Jeremy T. Bradley, and Uli Harder, editors, *Computer Performance Evaluation, Modelling Techniques and Tools 12th International Conference, TOOLS 2002, London, UK, April 14-17, 2002, Proceedings*, volume 2324 of *Lecture Notes in Computer Science*, pages 200–204. Springer, 2002.

[KNP11] Marta Kwiatkowska, Gethin Norman, and David Parker. Prism 4.0: Verification of probabilistic real-time systems. In *Computer Aided Verification: 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings 23*, pages 585–591. Springer, 2011.

[KPS08] Oleksiy Kurganskyy, Igor Potapov, and Fernando Sancho-Caparrini. Reachability problems in low-dimensional iterative maps. *Int. J. Found. Comput. Sci.*, 19(4):935–951, 2008.

[KSZ14] Joost-Pieter Katoen, Lei Song, and Lijun Zhang. Probably safe or live. In Thomas A. Henzinger and Dale Miller, editors, *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*, pages 55:1–55:10. ACM, 2014.

[KV01] Orna Kupferman and Moshe Y. Vardi. Model checking of safety properties. *Formal Methods Syst. Des.*, 19(3):291–314, 2001.

[Kwi07] Marta Z. Kwiatkowska. Quantitative verification: models techniques and tools. In Ivica Crnkovic and Antonia Bertolino, editors, *Proceedings of the 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2007, Dubrovnik, Croatia, September 3-7, 2007*, ESEC-FSE '07, pages 449–458, New York, NY, USA, 2007. ACM.

[KZ17] Felix Klein and Martin Zimmermann. How much lookahead is needed to win infinite games? *Logical Methods in Computer Science*, Volume 12, Issue 3, April 2017.

[Lam77] Leslie Lamport. Proving the correctness of multiprocess programs. *IEEE Trans. Software Eng.*, 3(2):125–143, 1977.

[Lam02] Leslie Lamport. Specifying systems: the tla+ language and tools for hardware and software engineers. 2002.

[Lan61] Rolf Landauer. Irreversibility and heat generation in the computing process. *IBM J. Res. Dev.*, 5(3):183–191, 1961.

[Lan64] P. J. Landin. The mechanical evaluation of expressions. *Comput. J.*, 6(4):308–320, 1964.

[LDL17] Yongming Li, Manfred Droste, and Lihui Lei. Model checking of linear-time properties in multi-valued systems. *Inf. Sci.*, 377:51–74, 2017.

[LMS22] Sylvain Lombardy, Victor Marsault, and Jacques Sakarovitch. *Awali, a library for weighted automata and transducers (version 2.3)*, 2022. Software available at http://vaucanson-project.org/Awali/2.3/.

[LP04] H. Leung and V. Podolskiy. The limitedness problem on distance automata: Hashiguchi's method revisited. *Theoretical Computer Science*, 310(1-3):147–158, 2004.

[LPRS03] Sylvain Lombardy, Raphael Poss, Yann Régis-Gianas, and Jacques Sakarovitch. Introducing VAUCANSON. In Oscar H. Ibarra and Zhe Dang, editors, *Implementation and Application of Automata, 8th International Conference, CIAA 2003, Santa Barbara, California, USA, July 16-18, 2003, Proceedings*, volume 2759 of *Lecture Notes in Computer Science*, pages 96–107. Springer, 2003.

[LPY97] Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a nutshell. *Int. J. Softw. Tools Technol. Transf.*, 1(1-2):134–152, 1997.

[LR10] Jay Ligatti and Srikar Reddy. A theory of runtime enforcement, with results. In Dimitris Gritzalis, Bart Preneel, and Marianthi Theoharidou, editors, *Computer Security - ESORICS 2010, 15th European Symposium on Research in Computer Security, Athens, Greece, September 20-22, 2010. Proceedings*, volume 6345 of *Lecture Notes in Computer Science*, pages 87–100. Springer, 2010.

[LSS+18] Martin Leucker, César Sánchez, Torben Scheffel, Malte Schmitz, and Alexander Schramm. Tessla: runtime verification of non-synchronized real-time streams. In Hisham M. Haddad, Roger L. Wainwright, and Richard Chbeir, editors, *Proceedings of the 33rd Annual ACM Symposium on Applied Computing, SAC 2018, Pau, France, April 09-13, 2018*, pages 1925–1933. ACM, 2018.

[LSS+19] Martin Leucker, César Sánchez, Torben Scheffel, Malte Schmitz, and Daniel Thoma. Runtime verification for timed event streams with partial information. In *International Conference on Runtime Verification*, pages 273–291. Springer, 2019.

[MB15] Menna Mostafa and Borzoo Bonakdarpour. Decentralized runtime verification of LTL specifications in distributed systems. In *2015 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2015, Hyderabad, India, May 25-29, 2015*, pages 494–503. IEEE Computer Society, 2015.

[McC60] John McCarthy. Recursive functions of symbolic expressions and their computation by machine, part I. *Commun. ACM*, 3(4):184–195, 1960.

[McM03] Kenneth L McMillan. Interpolation and sat-based model checking. In *Computer Aided Verification: 15th International Conference, CAV 2003, Boulder, CO, USA, July 8-12, 2003. Proceedings 15*, pages 1–13. Springer, 2003.

[MCW21a] Konstantinos Mamouras, Agnishom Chattopadhyay, and Zhifu Wang. Algebraic quantitative semantics for efficient online temporal monitoring. In Jan Friso Groote and Kim Guldstrand Larsen, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 27th International Conference, TACAS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 - April 1, 2021, Proceedings, Part I*, volume 12651 of *Lecture Notes in Computer Science*, pages 330–348. Springer, 2021.

[MCW21b] Konstantinos Mamouras, Agnishom Chattopadhyay, and Zhifu Wang. A compositional framework for quantitative online monitoring over continuous-time signals. In Lu Feng and Dana Fisman, editors, *Runtime Verification - 21st International Conference, RV 2021, Virtual Event, October 11-14, 2021, Proceedings*, volume 12974 of *Lecture Notes in Computer Science*, pages 142–163. Springer, 2021.

[MMMS21] Siddharth Mysore, Bassel Mabsout, Renato Mancuso, and Kate Saenko. Regularizing action policies for smooth control with reinforcement learning. In *IEEE International Conference on Robotics and Automation, ICRA 2021, Xi'an, China, May 30 - June 5, 2021*, pages 1810–1816. IEEE, 2021.

[MN04] Oded Maler and Dejan Nickovic. Monitoring temporal properties of continuous signals. In Yassine Lakhnech and Sergio Yovine, editors, *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems, Joint International Conferences on Formal Modelling and Analysis of Timed Systems, FORMATS 2004 and Formal Techniques in Real-Time and Fault-Tolerant Systems, FTRTFT 2004, Grenoble, France, September 22-24, 2004, Proceedings*, volume 3253 of *Lecture Notes in Computer Science*, pages 152–166. Springer, 2004.

[MO19] Jakub Michaliszyn and Jan Otop. Approximate learning of limit-average automata. In Wan J. Fokkink and Rob van Glabbeek, editors, *30th International Conference on Concurrency Theory, CONCUR 2019, August 27-30, 2019, Amsterdam, the Netherlands*, volume 140 of *LIPIcs*, pages 17:1–17:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[MO21] Jakub Michaliszyn and Jan Otop. Minimization of limit-average automata. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, pages 2819–2825. ijcai.org, 2021.

[MRA+17] Konstantinos Mamouras, Mukund Raghothaman, Rajeev Alur, Zachary G. Ives, and Sanjeev Khanna. Streamqre: modular specification and efficient evaluation of quantitative queries over streaming data. In Albert Cohen and Martin T. Vechev, editors, *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2017, Barcelona, Spain, June 18-23, 2017*, pages 693–708. ACM, 2017.

[NKF20] Mehran Alidoost Nia, Mehdi Kargahi, and Fathiyeh Faghih. Probabilistic approximation of runtime quantitative verification in self-adaptive systems. *Microprocess. Microsystems*, 72, 2020.

[NOT06] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Solving sat and sat modulo theories: From an abstract davis–putnam–logemann–loveland procedure to dpll (t). *Journal of the ACM (JACM)*, 53(6):937–977, 2006.

[NRZ+15] Chris Newcombe, Tim Rath, Fan Zhang, Bogdan Munteanu, Marc Brooker, and Michael Deardeuff. How amazon web services uses formal methods. *Communications of the ACM*, 58(4):66–73, 2015.

[NY20] Dejan Nickovic and Tomoya Yamaguchi. RTAMT: online robustness monitors from STL. In Dang Van Hung and Oleg Sokolsky, editors, *Automated Technology for Verification and Analysis - 18th International Symposium, ATVA 2020, Hanoi, Vietnam, October 19-23, 2020, Proceedings*, volume 12302 of *Lecture Notes in Computer Science*, pages 564–571. Springer, 2020.

[ORS16] Eric J Olson, James C Robinson, and Nicholas Sharples. Generalised cantor sets and the dimension of products. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 160, pages 51–75. Cambridge University Press, 2016.

[Pau17] Erik Paul. Monitor logics for quantitative monitor automata. In Kim G. Larsen, Hans L. Bodlaender, and Jean-François Raskin, editors, *42nd International Symposium on Mathematical Foundations of Computer Science, MFCS 2017,*

August 21-25, 2017 - Aalborg, Denmark, volume 83 of *LIPIcs*, pages 14:1–14:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[PH18] Doron Peled and Klaus Havelund. Refining the safety-liveness classification of temporal properties according to monitorability. In Tiziana Margaria, Susanne Graf, and Kim G. Larsen, editors, *Models, Mindsets, Meta: The What, the How, and the Why Not? - Essays Dedicated to Bernhard Steffen on the Occasion of His 60th Birthday*, volume 11200 of *Lecture Notes in Computer Science*, pages 218–234. Springer, 2018.

[PJT+17] Srinivas Pinisetty, Thierry Jéron, Stavros Tripakis, Yliès Falcone, Hervé Marchand, and Viorel Preoteasa. Predictive runtime verification of timed properties. *J. Syst. Softw.*, 132:353–365, 2017.

[PP18] Nir Piterman and Amir Pnueli. *Temporal Logic and Fair Discrete Systems*, pages 27–73. Springer International Publishing, Cham, 2018.

[PZ06] Amir Pnueli and Aleksandr Zaks. PSL model checking and run-time verification via testers. In Jayadev Misra, Tobias Nipkow, and Emil Sekerinski, editors, *FM 2006: Formal Methods, 14th International Symposium on Formal Methods, Hamilton, Canada, August 21-27, 2006, Proceedings*, volume 4085 of *Lecture Notes in Computer Science*, pages 573–586. Springer, 2006.

[QSCP22] Junyan Qian, Fan Shi, Yong Cai, and Haiyu Pan. Approximate safety properties in metric transition systems. *IEEE Trans. Reliab.*, 71(1):221–234, 2022.

[RC12] Grigore Rosu and Feng Chen. Semantics and algorithms for parametric monitoring. *Logical Methods in Computer Science*, 8, 2012.

[RCF+20] Alastair Reid, Luke Church, Shaked Flur, Sarah de Haas, Maritza Johnson, and Ben Laurie. Towards making formal methods normal: meeting developers where they are. *arXiv preprint arXiv:2010.16345*, 2020.

[RCR15] Giles Reger, Helena Cuenca Cruz, and David Rydeheard. Marq: monitoring at runtime with qea. In *Tools and Algorithms for the Construction and Analysis of Systems: 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015, Proceedings 21*, pages 596–610. Springer, 2015.

[Rén57] Alfréd Rényi. Representations for real numbers and their ergodic properties. *Acta Math. Acad. Sci. Hungar*, 8(3-4):477–493, 1957.

[Rey72] John C. Reynolds. Definitional interpreters for higher-order programming languages. In John J. Donovan and Rosemary Shields, editors, *Proceedings of the ACM annual conference, ACM 1972, 1972, Volume 2*, pages 717–740. ACM, 1972.

[RMT+04] Vlad Rusu, Hervé Marchand, Valéry Tschaen, Thierry Jéron, and Bertrand Jeannet. From safety verification to safety testing. In *Testing of Communicating Systems: 16th IFIP International Conference, TestCom 2004, Oxford, UK, March 17-19, 2004. Proceedings 16*, pages 160–176. Springer, 2004.

[RR15] Giles Reger and David Rydeheard. From first-order temporal logic to parametric trace slicing. In *Runtime Verification: 6th International Conference, RV 2015, Vienna, Austria, September 22-25, 2015. Proceedings*, pages 216–232. Springer, 2015.

[RST24] Lennard Reese, Rafael Castro G Silva, and Dmitriy Traytel. Timelymon: A streaming parallel first-order monitor. In *International Conference on Runtime Verification*, pages 150–160. Springer, 2024.

[SAA+21] N. Ege Saraç, Ömer Faruk Altun, Kamil Tolga Atam, Sertaç Karahoda, Kamer Kaya, and Hüsnü Yenigün. Boosting expensive synchronizing heuristics. *Expert Syst. Appl.*, 167:114203, 2021.

[SBAS04] Nisheeth Shrivastava, Chiranjeeb Buragohain, Divyakant Agrawal, and Subhash Suri. Medians and beyond: new aggregation techniques for sensor networks. In John A. Stankovic, Anish Arora, and Ramesh Govindan, editors, *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems, SenSys 2004, Baltimore, MD, USA, November 3-5, 2004*, pages 239–249. ACM, 2004.

[SBY06] Adam Silberstein, Rebecca Braynard, and Jun Yang. Constraint chaining: on energy-efficient continuous monitoring in sensor networks. In Surajit Chaudhuri, Vagelis Hristidis, and Neoklis Polyzotis, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, USA, June 27-29, 2006*, pages 157–168. ACM, 2006.

[SC82] A. Prasad Sistla and Edmund M. Clarke. The complexity of propositional linear temporal logics. In Harry R. Lewis, Barbara B. Simons, Walter A. Burkhard, and Lawrence H. Landweber, editors, *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA*, pages 159–168. ACM, 1982.

[Sch61] Marcel Paul Schützenberger. On the definition of a family of automata. *Inf. Control.*, 4(2-3):245–270, 1961.

[Sim94] I. Simon. On semigroups of matrices over the tropical semiring. *RAIRO-Theoretical Informatics and Applications*, 28(3-4):277–294, 1994.

[SS71] Dana S Scott and Christopher Strachey. *Toward a mathematical semantics for computer languages*, volume 1. Oxford University Computing Laboratory, Programming Research Group Oxford, 1971.

[SSSB21] Sandro Stucki, César Sánchez, Gerardo Schneider, and Borzoo Bonakdarpour. Gray-box monitoring of hyperproperties with an application to privacy. *Formal Methods Syst. Des.*, 58(1):126–159, 2021.

[Tea24] The Coq Development Team. The coq proof assistant, September 2024.

[Tof80] Tommaso Toffoli. Reversible computing. In J. W. de Bakker and Jan van Leeuwen, editors, *Automata, Languages and Programming, 7th Colloquium, Noordweijkerhout, The Netherlands, July 14-18, 1980, Proceedings*, volume 85 of *Lecture Notes in Computer Science*, pages 632–644. Springer, 1980.

[TR05] Prasanna Thati and Grigore Rosu. Monitoring algorithms for metric temporal logic specifications. *Electron. Notes Theor. Comput. Sci.*, 113:145–162, 2005.

[vG93] Rob J. van Glabbeek. The linear time - branching time spectrum II. In Eike Best, editor, *CONCUR '93, 4th International Conference on Concurrency Theory, Hildesheim, Germany, August 23-26, 1993, Proceedings*, volume 715 of *Lecture Notes in Computer Science*, pages 66–81. Springer, 1993.

[vG01] Rob J. van Glabbeek. The linear time - branching time spectrum I. In Jan A. Bergstra, Alban Ponse, and Scott A. Smolka, editors, *Handbook of Process Algebra*, pages 3–99. North-Holland / Elsevier, 2001.

[WDHR06] Martin De Wulf, Laurent Doyen, Thomas A. Henzinger, and Jean-François Raskin. Antichains: A new algorithm for checking universality of finite automata. In Thomas Ball and Robert B. Jones, editors, *Computer Aided Verification, 18th International Conference, CAV 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, volume 4144 of *Lecture Notes in Computer Science*, pages 17–30. Springer, 2006.

[WHK+13] Sigal Weiner, Matan Hasson, Orna Kupferman, Eyal Pery, and Zohar Shevach. Weighted safety. In Dang Van Hung and Mizuhito Ogawa, editors, *Automated Technology for Verification and Analysis - 11th International Symposium, ATVA 2013, Hanoi, Vietnam, October 15-18, 2013. Proceedings*, volume 8172 of *Lecture Notes in Computer Science*, pages 133–147. Springer, 2013.

[WYGG08] Chao Wang, Yu Yang, Aarti Gupta, and Ganesh Gopalakrishnan. Dynamic model checking with property driven pruning to detect race conditions. In Sung Deok Cha, Jin-Young Choi, Moonzoo Kim, Insup Lee, and Mahesh Viswanathan, editors, *Automated Technology for Verification and Analysis, 6th International Symposium, ATVA 2008, Seoul, Korea, October 20-23, 2008. Proceedings*, volume 5311 of *Lecture Notes in Computer Science*, pages 126–140. Springer, 2008.

[YHN24] Tomoya Yamaguchi, Bardh Hoxha, and Dejan Nickovic. RTAMT - runtime robustness monitors with application to CPS and robotics. *Int. J. Softw. Tools Technol. Transf.*, 26(1):79–99, 2024.

[ZLD12] Xian Zhang, Martin Leucker, and Wei Dong. Runtime verification with predictive semantics. In Alwyn Goodloe and Suzette Person, editors, *NASA Formal Methods - 4th International Symposium, NFM 2012, Norfolk, VA, USA, April 3-5, 2012. Proceedings*, volume 7226 of *Lecture Notes in Computer Science*, pages 418–432. Springer, 2012.