Masaryk University
Faculty of Informatics

Modern Probabilistic Verification

Habilitation Thesis

Jan Křetínský

Brno, 2018

To Zuzka

Abstract

Hardware and software verification is a mature technology, which has been adopted in many areas where correctness of behaviour is critical. In contrast, its younger *probabilistic* sibling still struggles at many basic points: the properties of interest are often hard to specify, verification engines do not scale well with the size of the system, and the products of the process are difficult to use. We show how machine learning, automata theory, and revisiting established concepts may help to address these issues, based on recent results.

Shrnutí

Hardwarová a softwarová verifikace se již etablovala v řade oblastí, kde je správné fungování systémů zcela klíčové. Naproti tomu její mladší, pravděpodobnostní odrůda stále zápolí se základními požadavky: není vždy jednoduché vyjádřit požadovanou vlastnost systému, použitelnost trpí přílišnou závislostí na velikosti systému a forma výsledků neodpovídá praktické potřebě. Tato habilitační práce ukazuje některé nedávno otevřené možnosti, jak strojové učení, teorie automatů a přehodnocení některých zaběhnutých pojmů a postupů můze v těchto ohledech pomoci.

Contents

PART I

COMMENTARY

Chapter 1

Introduction

In this chapter, we introduce the general topic of the thesis and outline its structure. While the subsequent chapters can be read independently, they all refer to the notions introduced in this chapter.

1.1 Probabilistic verification

Probabilistic systems are abundant in many areas ranging from telecommunication (randomized protocols), transportation (automotive, aerospace), operations research (queuing networks), biology (signalling pathways), to daily-life appliances (embedded software controllers), to name just a few.

Since many of the systems are safety-critical, we need to ensure their proper behaviour. To this end, we construct models of these systems and then analyze them with respect to, e.g., low consumption for resource-limited systems, high mean time to failure for dependable systems, or to gain understanding of complex behaviour of natural processes.

The process of *model checking* follows the general pattern depicted in Figure 1.1. The are two inputs to the procedure: a system and a property.

Firstly, the *systems* where probabilistic features are essential can be formalized in various ways, depending on further present features. In this thesis, we deal with the most fundamental models, namely

Markov chains (MC) [Mar06, Nor98] for fully stochastic systems,

Markov decision processes (MDP) [Bel57, Put94] capturing both stochasticity and non-determinism, either controllable or uncontrollable,

stochastic games (SG) [Sha53, Con90] with all the three present.

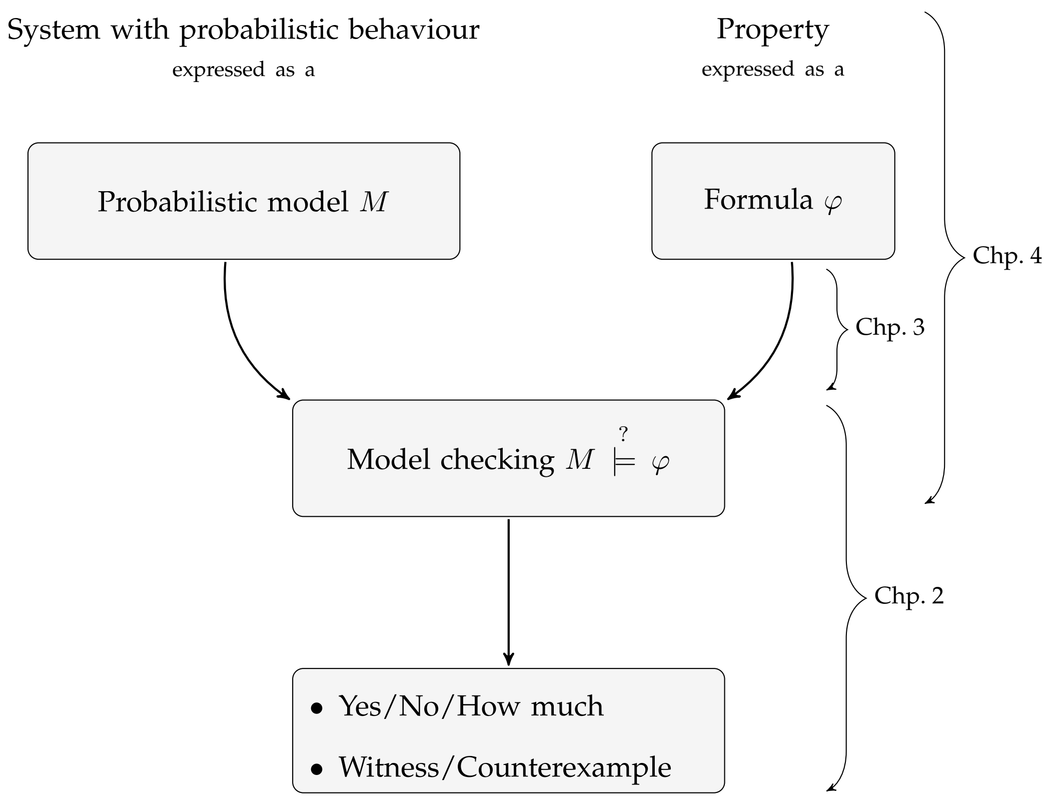


Figure 1.1: The general scheme of probabilistic model checking and the focus of the subsequent chapters

Besides, many richer formalism are defined in terms of MC and MDP, such as e.g. stochastic timed automata (STA) [BBB$^+$14] or probabilistic timed automata (PTA) [NPS13], designed to cope with additional timing issues. We will mostly focus on the basic models, such as MDP.

Secondly, the *properties* of interest range from the simplest properties of the system, e.g. **reachability** of a given state, to more complex properties defined by a quantitative structure over the system, e.g. **long-run average reward** [How60, Gil57], by a temporal formula, e.g. of **linear temporal logic (LTL)** [Pnu77], or by comparison to another system, e.g. **bisimulation** [Par81, LS89] or **distance** [CHR10]. Consequently, the property is usually formalized declaratively by a formula in a suitable logic or operationally by a behaviour of another system. We will mostly focus on the former.

As indicated in Figure 1.1, the model checking procedure combines and analyzes the input pair, yielding an answer to the question whether the

Table 1.1: Examples of properties of interest. The classic notions (line 1) have been extended to the probabilistic setting (line 2) and further generalized to more robust notions of quantitative probabilistic properties (line 3). Here $\mathbb{P}$ denotes probability, $\mathbb{E}$ expectation, R_i the reward obtained in the ith step, and $\mathsf{L(S)}$ the language recognized by S.

	LTL	average reward (mean payoff)	trace equivalence
classic	formula φ	$\mathsf{MP} := \liminf_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} R_i$	$\mathsf{L}(S_1) = \mathsf{L}(S_2)$
probabilistic	$\mathbb{P}[\varphi] = 1$	$\mathbb{E}[\mathsf{MP}] \geq t$	$\forall L : \mathbb{P}_1[L] = \mathbb{P}_2[L]$
quantitative pr.	$\mathbb{P}[\varphi] \geq p$	$\mathbb{P}[\mathsf{MP} \geq t] \geq p$	$\forall L : \lvert\mathbb{P}_1[L] - \mathbb{P}_2[L]\rvert < d$

model satisfies the formula. However, for probabilistic systems, the Boolean notions of satisfaction and equivalence are not always useful. Rather we need to refine the notions into truly quantitative ones, such as the *extent* of satisfaction, e.g. "*with 95% chance* the long-run average reward is between 0 and 42". For instance, even highly safety-critical systems such as nuclear plants, with each hardware component failing with certain probability, do not satisfy the safety properties, but with some (preferably high) probability. Computing this probability is the task of *quantitative probabilistic model checking*. Similarly, the probabilities of failures of the components are only empirically estimated and the slightest imprecision in the estimate may result in systems being or not being equivalent. Instead, we prefer to measure *how much they differ*. This can be captured by the quantitative notion of *distance*. Several examples of popular properties are depicted in Table 1.1.

Further, the answer may be *documented* by a witness, e.g. a motion-planning strategy to be implemented, or a counterexample, e.g. scheduling leading to violation of mutual exclusion. This is crucial for the practical applicability of model checking and the core point of controller synthesis.

The approach to the analysis depends not only on the already discussed types of the two inputs and of the output, but also on

the granted knowledge of the model ranging from

- full quantitative information, white-box models (assumed in most of the literature) to
- qualitative one (without the exact quantities) to
- black-box models, which can only be simulated and their behaviour observed in runtime, and

the required guarantee on the result ranging from

- precise/optimal to
- ε-precise/ε-optimal (for a known or given error bound ε)
- to confidence intervals typical in statistical model checking [YS02, ISOLA'16] and probably approximately correct results (PAC) typical in learning [Val84, SLW$^+$06] to,
- best effort, which is mostly out of scope of the thesis.

1.2 Focus and structure of the thesis

The thesis reports on recent[(*)] progress in making probabilistic verification more efficient, in particular *more scalable* despite the state-space-explosion problem, and *more usable* in that (i) the input formula reflects the desired property more accurately, and (ii) the output artefact (counterexample or the produced controller) are easier to understand, debug or implement.

The content can be classified into three streams and we outline it in the order we explained Figure 1.1:

- Chapter 2 discusses how inexact approaches based on simulations and machine learning may improve both exact analysis *and* the quality of its output.
- Chapter 3 focuses on LTL and how to transform its formulae into different types of automata, allowing for their efficient analysis for each particular setting.
- Chapter 4 handles more complex properties and their extensions, combinations, and alternative interpretations.

The main novel aspects are thus

- bridging some of the gaps between *formal* and *practically used methods*,
- innovations in *automata theory*, leading to practical improvements, and
- identification of new *specification concepts* and their verification procedures, respectively.

The second part then lists some of the papers on which the discussed results are based.

(∗). I.e., since obtaining the second Ph.D. in mid-2014.

Chapter 2

Learning to Control

In this chapter, we explain how imprecise techniques based on simulations and machine learning can enhance traditional verification and controller-synthesis techniques in several ways:

- improving scalability without compromising the result,
- improving scalability for relaxed statistical guarantees, particularly for black-box systems where stronger guarantees are not possible anyway,
- improving structure and size of the produced results.

.

2.1 State of the art

Verification offers a range of methods for solving various classes of probabilistic verification problems.

2.1.1 Numerical verification

For finite systems, graph algorithms are quite efficient, e.g. [CY95] for MC and MDP, but only work for purely qualitative questions. In the generally quantitative case (quantitative interpretation in line 2 of Table 1.1, or quantitative objectives such as mean payoff in the middle column), numerical methods are usually used, most notably

(i) dynamic programming [Put94], such as *value iteration* (VI), used in the most used probabilistic model checker PRISM [KNP02], or *policy/strategy iteration* (SI), or

(ii) linear programming (LP) as used in e.g. DiVinE [BBC+08].

On the one hand, LP provides precise results. On the other hand, it is slow for MDP and not applicable to SG. Since the repetitive evaluation of strategies in SI is often slow in practice, VI is usually preferred. For instance, the most used probabilistic model checker PRISM [KNP02] and its branch PRISM-Games [CFK+13] use VI for MDP and SG as the default option, respectively. However, while SI is in principle a precise method, VI is an approximative method, which converges only in the limit. Unfortunately, there was no known stopping criterion for VI. Consequently, there were no guarantees on the results returned in finite time. Therefore, current tools stop when the difference between the two most recent approximations is low, and thus may return arbitrarily imprecise results [HM14]. A stopping criterion has been developed in [HM14] and published in parallel with our [ATVA'14a], for details see Section 2.2.1.

Continuous systems are often transformed into finite systems. The methods typically include either some form of *discretization* with derived error bounds, as in e.g. MRMC [KZH+11] for continuous-time MDPs, or use more structured finite abstractions, such as regions and zones for (probabilistic extensions of) timed automata, as in e.g. Uppaal [BDL+06, DLL+15] or PRISM [KNP02].

All of these methods provide guarantees on the results. However, the price to pay for that is non-trivial complexity with respect to the size of the system and the property. Furthermore, due to the curse of dimensionality the systems grow exponentially with respect to the number of variables and components involved. Consequently, the real systems mostly cannot be treated directly. This issue is under a heavy attack of many techniques:

- Firstly, there are *symbolic* techniques avoiding explicit treatment of the whole state space, e.g. [CHJS11, BKH99, ZSF12], and combinations of explicit and symbolic techniques [WBB+10, BBR14].

- Secondly, there are *model-transformation* techniques:

  - *Compositional* techniques aim to analyse parts of the systems separately and combine the results to infer properties of the whole, e.g. [CDL+10, DH11, BKW14, CCD15] or our [HKK13].
  - *Abstraction* techniques merge states, factoring out information irrelevant for the satisfaction of the particular property [KKNP10, HWZ08, HHWZ10] or our [SKC+15].

- Dual to abstractions, there are *reduction* techniques, delimiting and analysing only subsystems, thus considering only some behaviours. An example of a safe technique is the partial order (or symmetry) reduction [BGC04, GDF09]. Unfortunately, there are not too many safe reduction techniques. In contrast, non-safe reduction techniques are extensively used in practice. For instance, stochastic simulation is a very useful debugging technique. It can disprove properties and find deficiencies of the system. However, it cannot be simply used to prove properties and assure the system correctness.

2.1.2 Statistical model checking

Verification of MC, MDP, and related systems traditionally relies on numerical approaches. However, numerical analysis of the whole system is often inapplicable in practice:

(i) when the system is too large due to state space explosion, or

(ii) when the exact transitions are unknown (*black-box systems*).

In such cases, *statistical* approaches and simulation form a powerful alternative. The statistical approach typically consists in

1. observing (finitely many finitely long) simulation runs,
2. analysis of each run,
3. inferring properties of the system from statistics on the results of the analysis.

Since simulations can often be done very fast, this approach often scales very well. The increased performance, however, comes at a cost of providing only *probabilistic guarantees* on the result, i.e., the result of the analysis is correct only with probability $1 - \varepsilon$ for a user-given ε. In many contexts this is no limitation since ε can be very small as the algorithms scale well, so that the uncertainty is negligible. This may be particularly the case when the model itself is validated with respect to the real system only with some certainty.

Statistical model checking (SMC) [YS02] has been successfully applied to various biological [JCL+09, PGL+13], hybrid [ZPC10, DDL+12, EGF12, Lar12] or cyber-physical [BBB+10, CZ11, DDL+13] systems and there is a substantial tool support available [JLS12, BDL+12b, BCLS13, BHH12].

2.1.2.1 Statistical model checking for MCs

Statistical model checking (SMC) [YS02] of Markov chains refers to algorithms with the following specification:

Specification of Markov chains statistical model checking

Input:

- a finite black-box MC $\mathcal{M}$ (i.e., access to any desired finite number of sampled simulation paths of any desired finite lengths)
- a linear property φ
- a threshold probability p
- an indifference region $\varepsilon > 0$
- two error bounds $\alpha, \beta > 0$
- possibly some characteristics of $\mathcal{M}$ from Table 2.1

Output: if $\mathbb{P}[\mathcal{M} \models \varphi] \geq p+\varepsilon$, return YES with probability at least $1-\alpha$;
if $\mathbb{P}[\mathcal{M} \models \varphi] \leq p-\varepsilon$, return NO with probability at least $1-\beta$.

Bounded and Unbounded Properties Most of the previous efforts in SMC has focused on the analysis of properties with bounded horizon [YS02, SVA04, YKNP06, JCL$^+$09, JLS12, BDL$^+$12b]. For such bounded properties (e.g. state r is reached with probability at most 0.5 in the first 1000 steps), statistical guarantees can be obtained in a completely black-box setting, where execution runs of the system can be observed, but no other information is available. Unbounded properties (e.g. state r is reached with probability at most 0.5 in any number of steps) are significantly more difficult, as a stopping criterion is needed when generating a potentially infinite execution run, and some information about the system is necessary for providing statistical guarantees. Table 2.1 presents an overview of the assumptions for the statistical analysis of unbounded properties, detailed below.

SMC of unbounded properties, usually "unbounded until" properties, was first considered in [HLMP04] and the first approach was proposed in [SVA05], but observed incorrect in [HJB$^+$10]. Notably, in [YCZ10] two approaches are described. The *first approach* proposes to terminate sampled paths at every step with some probability p_{term} and re-weight the result accordingly. In order to guarantee the asymptotic convergence of this method,

Table 2.1: Statistical approaches organised by (i) the class of verified linear properties, and (ii) by the required information about the Markov chain, where p_{min} is the minimal transition probability, $|S|$ is the number of states, and λ is the second largest eigenvalue of the chain. Further, $\Diamond$ denotes the (unbounded) reachability and $\mathbf{U}$ the (unbounded) until of LTL.

property	no info	p_{min}	$\lvert S\rvert, p_{\mathsf{min}}$	λ	topology
bounded	e.g. [YS02, SVA04]				
$\Diamond, \mathbf{U}$	$\times$	[TACAS'16]	[ATVA'14a]	[YCZ10]	[YCZ10, HJB$^+$10]
LTL, MP	$\times$	[TACAS'16]	[ATVA'14a]		

the second eigenvalue λ of the chain must be computed, which is as hard as the verification problem itself. It should be noted that the method provides only asymptotic guarantees as the width of the confidence interval converges to zero. The correctness of [LP08] relies on the knowledge of the second eigenvalue λ, too. The *second approach* of [YCZ10] requires the qualitative knowledge, i.e. the chain's topology, which is used to transform the chain so that all potentially infinite paths are eliminated. In [HJB$^+$10], a similar transformation is performed, again requiring knowledge of the topology. The (pre)processing of the state space required by the topology-aware methods, as well as by traditional numerical methods for Markov chain analysis, is a major practical hurdle for large (or unknown) state spaces. Another approach, limited to ergodic Markov chains, is taken in [RP09], based on coupling methods. There are also extensions of SMC to timed systems [DLL$^+$15].

2.1.2.2 Statistical model checking for MDPs

The development of statistical model checking techniques for probabilistic models with *nondeterminism*, such as MDPs, has only been treated quite recently. In [BFHH11], properties are analysed for MDPs with spurious non-determinism, where the way it is resolved does not affect the desired property. In the case with general non-determinism, one approach is to give the non-determinism a probabilistic semantics, e.g., using a uniform distribution instead, as for timed automata in [DLL$^+$11a, DLL$^+$11b, Lar13]. Others [LP12, HMZ$^+$12, ATVA'14a] aim to quantify over all strategies and produce an ϵ-optimal strategy. The works of [HMZ$^+$12] and [LP12] deal with the problem in the setting of bounded and discounted (and for the purposes of approximation thus bounded) properties, respectively. In the former work, candidates for optimal strategies are generated and gradually improved, but “at any given point we cannot quantify how close to optimal the candidate scheduler is” (cited from [HMZ$^+$12]) and the algorithm “does not in general converge to the true optimum” (cited from [LST14]). Further, [LST14] randomly samples (compact representation of) strategies, but again focuses only on (time-)bounded properties.

There are also various practically efficient heuristics that, however, provide none or very weak guarantees, often based on some form of learning [BT00, LL08, WT16, TT16, AY17, BBS08]. Even for MDP, the first PAC algorithm (limited to discounted reward) has been given in [SLW$^+$06].

2.1.3 Strategy Representation

For systems with non-determinism, both counterexamples as well as witnesses are given as strategies, resolving the non-determinism. Representing the resulting strategy compactly is important in both cases since either (i) it needs to be implemented as a controller and must be simple enough, or (ii) it is a counterexample when trying to prove a property for all strategies and then the corresponding bug needs to be understood and fixed. There are several different classes of data structures and algorithms to represent strategies.

Firstly, in artificial intelligence, compact (factored) representations of MDP structure have been developed using dynamic Bayesian networks [BDG95, KK99], probabilistic STRIPS [KHW94], algebraic decision diagrams [HSAHB99], and also decision trees [BDG95]. Formalisms used to represent MDPs can, in principle, be used to represent values and strategies as well. In particular, variants of decision trees are probably the most used [BDG95, CK91, KP99]. For a detailed survey of compact representations see [BDH99].

Secondly, in the context of verification, MDPs are often represented using variants of (MT)BDDs [dAKN$^+$00, HKN$^+$03, MP04], and strategies by BDDs [WBB$^+$10].

Thirdly, [AL09] uses a directed on-the-fly search to compute sets of most probable diagnostic paths. The notion of paths encoded as AND/OR trees has also been studied in [LL13] to represent probabilistic counter-examples visually as fault trees, and then derive causal relationship between events. Further, [WJV$^+$13, DJW$^+$14] compute a smallest set of guarded commands (of a PRISM-like language) inducing a violating subsystem, but, unlike other methods, does not provide a compact representation of actual decisions needed to reach an erroneous state; moreover, there is not always a command-based counterexample.

Finally, *decision trees* have been used in connection with real-time dynamic programming and reinforcement learning to represent the learned approximation of the value function [BD96, Pye03]. Learning a compact decision tree representation of a strategy has been investigated in [SLT10] for the case of body sensor networks with discounted objectives.

2.2 Contributions

2.2.1 Numerical verification

In this section, we employ simulations and reinforcement learning to speed up dynamic-programming techniques, such as VI. The structure of the approach is as follows. Firstly, while VI traditionally approximates the value from below, we show how to modify it so that it approximates the value from above, too. Consequently, we obtain a stopping criterion for VI: when the difference between the under- and over-approximation is smaller than the desired precision, we can stop. Secondly, we show how the lower and upper bounds can be utilized to derive guarantees on generally unreliable methods based on simulation and learning, resulting in an efficient reduction technique.

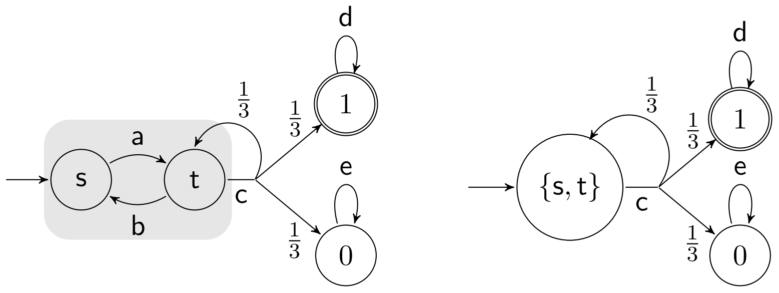
Stopping criterion for VI Deriving over-approximating VI sequence converging to the value is non-trivial already for MDP with the reachability objective. The reason is that the greatest fixpoint of the VI operator (the greatest solution to Bellman equations) can be strictly greater than the actual value. This is illustrated in Figure 2.1.

In [ATVA'14a] we show that when so-called *end components (EC)* [DA97] are abstracted into a single state then there is only a single fixpoint of the VI operator. Hence both sequences converge to the same value. Consequently, the over-approximating sequence on such modified system induces an over-approximating sequence on the original system that converges to the actual original value. This has been independently discovered also in [HM14] a few months after the first submission of [ATVA'14a].

In [CAV'18b] we extend this approach to SG. Here end components cannot be abstracted into single states since the values of the individual states are different. Instead, we consider varying temporary abstractions depending on the current approximations. This is illustrated in Figure 2.2.

In [CAV'17] we extend the stopping criterion for MDP with reachability of [ATVA'14a] to MDP with long-run average reward.

Asynchronous value iteration There are variants of VI, where we apply the VI operator "asynchronously", i.e. with varying frequencies for different states. For such asynchronous methods, nothing is known about the



i	$L_i(\mathsf{t}) = L_i(\{\mathsf{s},\mathsf{t}\})$	$U_i(\mathsf{t})$=$U_i(\mathsf{s})$	$U_i(\{\mathsf{s},\mathsf{t}\})$
0	0	1	1
1	$\frac{1}{3}$	1	$\frac{2}{3}$
2	$\frac{4}{9}$	1	$\frac{5}{9}$
3	$\frac{13}{27}$	1	$\frac{14}{27}$

Figure 2.1: Example illustrating the iterations of lower (L) and upper (U) bounds for maximum reachability of the double circled 1. Left: An MDP (as special case of SG) where the greatest fixpoint $\lim_{i\to\infty} U_i(\mathsf{t}) = 1$ is different from the value ($\frac{1}{2}$) due to the grayed EC $\{\mathsf{s},\mathsf{t}\}$. This is due to the fact that $U_i(\mathsf{t})$ depends on $U_i(\mathsf{s})$ and vice versa and, intuitively, s and t mutually suggest the possibility of the value being up to 1 although this illusion is not based on real paths to the target of this measure. Right: The same MDP where the EC is "collapsed" into a single state, ensuring the convergence $\lim_{i\to\infty} U_i(\{\mathsf{s},\mathsf{t}\}) = \frac{1}{2}$. Bottom: The approximations illustrating the non/convergence in the first few steps.

speed of convergence. Yet, since we can guarantee rigorous lower and upper bounds, we know the current precision of the approximations. Therefore, we can apply techniques from reinforcement learning (Q-learning) [Wat89] or probabilistic planning (bounded real-time dynamic programming, BRTDP) [MLG05] and still obtain results with guaranteed precision.

Moreover, since the frequencies can be arbitrary, the result is often output without ever examining some states. This is illustrated in Figure 2.3. Altogether, this approach allows us to focus on the most important part of the system and ignore the rest. Since the former can be orders of magnitude smaller, this reduction technique is a useful tool to fight the state space explosion problem, for illustration see Figure 2.3. For examples of reductions in the size of the considered part of the state space, see Table 2.2.

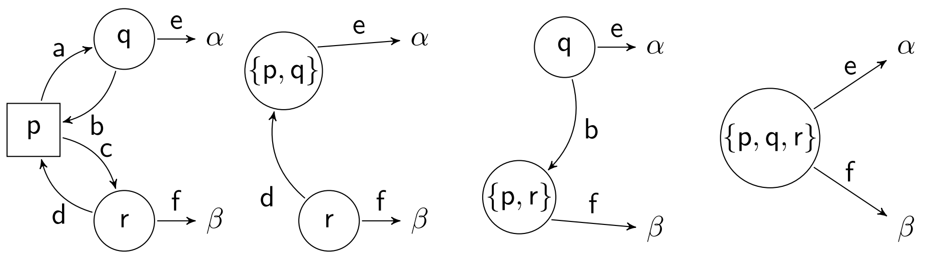


Figure 2.2: Left: Example of SG where states of an EC have different values. The Greek letters on the leaving arrows denote the values of the actions. Round and square states belong to the maximizer and minimizer, respectively. Right three figures: Correct collapsing in cases where $\alpha < \beta$, $\alpha > \beta$, and $\alpha = \beta$, respectively. In contrast to MDP, some actions of the EC leaving the collapsed part have to be removed.

Table 2.2: Reductions in the considered part of the state space on examples from the PRISM Benchmark Suite [KNP12]. The two columns display the size of the whole state space and the size of the part explored by our approach in order to approximate the value with precision 10^{-6}.

Example	Number of states visited by	
	PRISM [KNP02]	our [ATVA'14a]
zeroconf	4,427,159	977
wlan	5,007,548	1,995
firewire	19,213,802	32,214
mer	26,583,064	1,950

We provide and experimentally test this approach for MDP with reachability and LTL in [ATVA'14a], for MDP with long-run average reward in [CAV'17] and an SI variant in [ATVA'17], and for SG with reachability in [CAV'18b]. We extend the approach to continuous time in [ATVA'18b]. We also consider a hybrid approach combining BRTDP and Monte Carlo tree search in [ISOLA'18].

The basic algorithm of [HM14] is implemented in PRISM [BKL$^+$17] and the learning approach of [ATVA'14a] in STORM [DJKV17] and together with the extensions in our distribution of PRISM$^{(*)}$. The extension for SG where the interleaving of players is severely limited (every EC belongs to one player only) is discussed in [Ujm15].

(∗). Accessible at `https://gitlab.lrz.de/i7/prism`

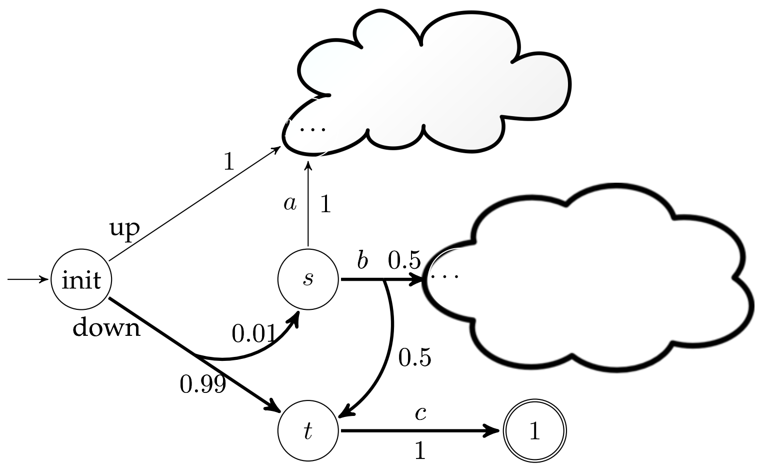


Figure 2.3: Example of an MDP where the parts of the system in the clouds can be ignored if the required precision is 0.005. Choosing the thick actions is guaranteed to result in an 0.005-optimal strategy, no matter how other non-determinism is resolved.

2.2.2 Statistical model checking

In this section, we employ simulations and reinforcement learning to analyze (partially) unknown systems.

2.2.2.1 Statistical model checking for MC

We present SMC algorithms for unbounded properties that, however, do not require much knowledge of the system. They are based on detecting that a simulation run reached a so-called bottom strongly connected component (BSCC), i.e. an EC of the MC regarded as an MDP. Then and only then we can deduce what the rest of the infinite run will be like and can thus terminate the run. Moreover, this also implies that such algorithms can be applied not only to reachability and unbounded-until properties, but can also be easily extended to LTL or mean payoff.

In [ATVA'14a] a priori bounds for the length of execution runs are calculated from the minimum transition probability p_{min} and the number of states $|S|$ only. After a long enough trace, we can deduce we are in a BSCC with very high probability. The length of the trace can be bounded (for a given confidence) by $|S|$ and p_{min}. Indeed, if there is a way out of the BSCC it is sufficient to take a path of length S, which has probability at least $p_{\mathsf{min}}^{|S|}$.

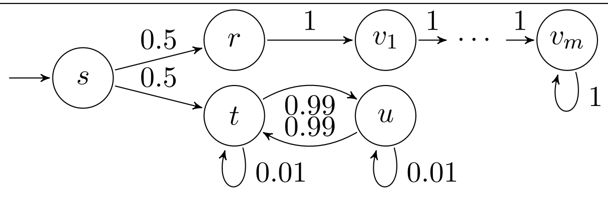


Figure 2.4: Example of a Markov chain with non-trivial BSCC detection

However, without taking execution information into account, these bounds are exponential in the number of states and highly impractical, as illustrated in the example below.

In [TACAS'16] we further improve on this idea and declare that we have reached a BSCC if the same states are repeated for long enough time. Since we observe the visited states we do not need the size of the state space $|S|$ as a bound, but only p_{min}. This is the first SMC algorithm that uses information obtained from execution prefixes. We illustrate and compare the two approaches on the following example.

Example. Consider the property of reaching state r in the Markov chain depicted in Figure 2.4. While the execution runs reaching r satisfy the property and can be stopped without ever entering any v_i, the finite execution paths without r, such as $stuttutuut$, are inconclusive. In other words, observing this path does not rule out the existence of a transition from, e.g., u to r, which, if existing, would eventually be taken with probability 1. This transition could have arbitrarily low probability, rendering its detection arbitrarily unlikely, yet its presence would change the probability of satisfying the property from 0.5 to 1. However, knowing that if there exists such a transition leaving the set, its transition probability is at least $p_{\mathsf{min}} = 0.01$, we can estimate the probability that the system is stuck in the set $\{t, u\}$ of states. Indeed, if existing, the exit transition was missed at least four times during the execution above, no matter whether it exits t or u. Consequently, the probability that there is no such transition and $\{t, u\}$ is a BSCC is at least $1 - (1 - p_{\mathsf{min}})^4$.

This means that in the approach of [TACAS'16], in order to get 99% confidence that $\{t, u\}$ is a BSCC, we only need to see both t and u around 500 times on a run, since $1 - (1 - p_{\mathsf{min}})^{500} = 1 - 0.99^{500} \approx 0.993$. This is in stark contrast to a priori bounds that provide the same level of confidence, such as the $(1/p_{\mathsf{min}})^{|S|} = 100^{\mathcal{O}(m)}$ runs required by [ATVA'14a], which is infeasible for large m of our example. In contrast, the performance of the method [TACAS'16] is independent of m. △

Consequently, many execution runs can be stopped quickly. Moreover, since the number of execution runs necessary for a required confidence level is independent of the size of the state space, it is not very large even for highly confident results (think of opinion polls). Altogether, it efficiently fights the state space explosion for systems where strongly connected components are not too large.

2.2.2.2 Statistical model checking for MDP

Further, [ATVA'14a] is also the first to consider SMC for MDP with unbounded properties. It explores (similarly to [HMZ+12]) the opportunities offered by learning-based methods, as used in fields such as planning or reinforcement learning. The algorithm assumes information limited to $|S|$ and p_{min} and is based on *delayed Q-learning* (DQL) [SLW+06]. Like in the numeric algorithm above, it maintains both lower and upper bounds on the result and gradually improves them. Contrary to the numeric algorithm, these bounds are not guaranteed to be correct, but only probably approximately correct (PAC) since there is a non-zero probability that the empirical estimates of the behaviour are significantly incorrect. However, this probability can be set arbitrarily close to 0.

The crucial steps of [ATVA'14a] are (1) modifying the DQL algorithm with PAC guarantees of [SLW+06] from the discounted setting to the undiscounted setting, but where terminating states are reached almost surely, and (2) lifting this to general MDPs with ECs, where terminating states may not be reached. This relies on simulations and the detection of end components on the fly. This technique extends also to LTL objectives and thus also to both maximum and minimum probabilities.

Further, we consider MDP with qualitative knowledge only and treat a combination of ω-regular objectives and long-run average reward in [CONCUR'18a].

2.2.3 Strategy representation

In this section, we employ simulations and decision-tree learning to post-process strategies so that they are (i) more understandable and (ii) more resource-efficient with respect to memory and time, i.e. smaller and faster to run. We stipulate that the size is the basic measure and can serve as a proxy for the others: simplicity and execution speed.

In [CAV'15a], we propose three steps to obtain smaller strategy representation. Each of them has a positive effect on the resulting size.

1. *Obtaining a (possibly partially defined and non-deterministic) ε-optimal strategy.* The ε-optimal strategies produced by standard meth-

ods, such as VI of PRISM [KP13], may be too large to compute and overly specific. Firstly, as argued in [ATVA'14a], typically only a small fraction of the system needs to be explored in order to find an ε-optimal strategy, whereas most states are reached with only a very small probability. Without much loss, the strategy may not be defined there. For example, in the MDP depicted in Figure 2.3, the decision in s (and the clouds) are almost irrelevant for the overall probability of reaching 1 from init. Such a partially defined strategy can be obtained using [ATVA'14a].

2. *Identifying important parts of the strategy.* Given a strategy, we define a concept of *importance* of a state s for reaching $goal$ is defined by $\mathbb{P}[\Diamond s \mid \Diamond goal]$. Let us shed some light on this definition. Observe that only a fraction of states can be reached while following the strategy, and thus have positive importance. On the unreachable states, with zero importance, the definition of the strategy is useless. For instance, in the previous example, also the upper cloud was partially explored in order to find out whether it is better to take action up or $down$. However, if the resulting strategy is to use $down$ and b, the information what to do in the upper cloud is useless. In addition, we consider the lower cloud to be of zero importance, too, since its states are never reached on the way to target and thus cannot be utilized. Furthermore, apart from ignoring states with zero importance, it is desirable to partially ignore decisions that are unlikely to be made (in less important states such as s), and in contrast, stress more the decisions in important states likely to be visited (such as $init$). The crucial notion of importance is obviously not computed, but only estimated *statistically by simulating* the system under the given strategy.

3. *Data structures for compact representation of strategies.* The explicit representation of a strategy by a table of pairs (state, action to play) results in a huge amount of data since the systems often have millions of states. Therefore, a symbolic representation by binary decision diagrams (BDD) looks as a reasonable option. However, there are several drawbacks of using BDDs. Firstly, due to the bit-level representation of the state-action pairs, the resulting BDD is not very readable. Secondly, it is often still too large to be understood by human, for instance due to a bad ordering of the variables. Thirdly, it cannot quantitatively reflect the differences in the importance of states. Of course, we can store decisions in states with importance

Table 2.3: Representation of strategies for several examples of PRISM Benchmark Suite [KNP12]. On the left, we display the size of the state space and the maximum reachability probability. On the right, we display the sizes of the representations using explicit listings of sets, BDD, decision trees, and the relative precision of the strategy induced by the generated tree. The trees are pruned as much as possible to still keep this error below 1%. On the mer benchmark, PRISM mems out while outputting the strategy. The line below shows the case with a partial strategy computed by our method [BCC+14] discussed above, for precision 10^{-6}.

Example	States	Value	Explicit	BDD	DT	Rel.err(DT) %
firewire	481,136	1.0	479,834	4233	1	0.0
investor	35,893	0.958	28,151	783	27	0.886
mer	1,773,664	0.200016	——— MEM-OUT ———			
			1887	619	13	0.00014
zeroconf	89,586	0.00863	60,463	409	7	0.106

above a certain threshold. However, we obtain much smaller representations and solve all the three issues if we allow more variability and reflect the whole quantitative information by decision-tree learning, using entropy.

We demonstrate the efficiency of the approach for MDP in [Cav'15a], see Table 2.3. We also give examples how reasons for present bugs can be read off from the decision trees.

Further, we modify the approach for non-stochastic games in [Tacas'18]. Interestingly, when applied to parametrized examples, the strategies for different values of the parameters are sometimes so similar that a generic solution can be read off. This suggests a potential of this technique for parametrized synthesis.

2.3 Contributed papers and activities

[Atva'14a] Tomáš Brázdil, Krishnendu Chatterjee, Martin Chmelík, Vojtěch Forejt, Jan Křetínský, Marta Z. Kwiatkowska, David Parker, and Mateusz Ujma. Verification of Markov decision processes using learning algorithms. In *ATVA*, volume 8837 of *LNCS*, pages 98–114. Springer, 2014.
Attached in Part 2 of the thesis.

[CAV'15a] Tomáš Brázdil, Krishnendu Chatterjee, Martin Chmelík, Andreas Fellner, and Jan Křetínský. Counterexample explanation by learning small strategies in Markov decision processes. In *CAV (1)*, volume 9206 of *LNCS*, pages 158–177. Springer, 2015.
Attached in Part 2 of the thesis.

[TACAS'16] Przemyslaw Daca, Thomas A. Henzinger, Jan Křetínský, and Tatjana Petrov. Faster statistical model checking for unbounded temporal properties. In *TACAS*, volume 9636 of *LNCS*, pages 112–129. Springer, 2016.
Journal version published in *ACM Transaction on Computational Logic*, 18(2):12:1–12:25, 2017.
Attached in Part 2 of the thesis.

[ISOLA'16] Jan Křetínský. Survey of statistical verification of linear unbounded properties: Model checking and distances. In *ISoLA (1)*, volume 9952 of *LNCS*, pages 27–45, 2016.

[CAV'17] Pranav Ashok, Krishnendu Chatterjee, Przemyslaw Daca, Jan Křetínský, and Tobias Meggendorfer. Value iteration for long-run average reward in Markov decision processes. In *CAV (1)*, volume 10426 of *LNCS*, pages 201–221. Springer, 2017.

[ATVA'17] Jan Křetínský and Tobias Meggendorfer. Efficient strategy iteration for mean payoff in Markov decision processes. In *ATVA*, volume 10482 of *LNCS*, pages 380–399. Springer, 2017.

[TACAS'18] Tomás Brázdil, Krishnendu Chatterjee, Jan Křetínský, and Viktor Toman. Strategy representation by decision trees in reactive synthesis. In *TACAS (1)*, volume 10805 of *LNCS*, pages 385–407. Springer, 2018.

[CAV'18b] Edon Kelmendi, Julia Krämer, Jan Křetínský, and Maximilian Weininger. Value iteration for simple stochastic games: Stopping criterion and learning algorithm. In *CAV (1)*, volume 10981 of *LNCS*, pages 623–642. Springer, 2018.
Attached in Part 2 of the thesis.

[DAGSTUHL'18] Nils Jansen, Joost-Pieter Katoen, Pushmeet Kohli, and Jan Křetinsky (eds.). Machine learning and model checking join forces (Dagstuhl seminar 18121). *Dagstuhl Reports*, 8(3):74–93, 2018.

[CONCUR'18a] Jan Křetínský, Guillermo A. Pérez, and Jean-François Raskin. Learning-based mean-payoff optimization in an unknown MDP under ω-regular constraints. In *CONCUR*, volume 118 of *LIPIcs*, pages 32:1–32:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.

[ATVA'18b] Pranav Ashok, Yuliya Butkova, Holger Hermanns, and Jan Křetínský. Continuous-time Markov decisions based on partial exploration. In *ATVA*, volume 11138 of *LNCS*, pages 317–334. Springer, 2018.

[ISOLA'18] Pranav Ashok, Tomáš Brázdil, Jan Křetínský, and Ondřej Slámečka. Monte Carlo tree search for verifying reachability in Markov decision processes. In *ISoLA*. To appear, 2018.

Activities Combining learning and formal methods is currently a hot topic, discussed at venues for establishing new directions, such as Dagstuhl seminars. The author has co-organized one on this topic [DAGSTUHL'18] and presented it in five other Dagstuhl seminars.

In 2018, the author has given invited talks on the topic at *Logic and Learning* workshop at the Alan Turing Institute, FNRS seminar at Brussels Free University, University of Twente, or FOPSS summer school *Logic and Learning* at FLoC in Oxford.

On this topic, the author has obtained a German Research Foundation grant *Statistical Unbounded Verification* in 2017 and twice passed to the second round of ERC Starting Grant application (scoring A and B).

Chapter 3

From LTL to Automata

In this chapter, we explain how automata theory can improve scalability of model checking for complex properties.

3.1 State of the art

Automata-theoretic approach [VW86] is a key technique for verification and synthesis of systems with linear-time specifications, such as formulae of linear temporal logic (LTL) [Pnu77]. It proceeds in two steps: first, the formula is translated into a corresponding automaton; second, the product of the system and the automaton is further analyzed. For an instantiation of the framework, see Figure 3.2. The size of the automaton is important as it directly affects the size of the product and thus largely also the analysis time, particularly for deterministic automata and probabilistic model checking in a very direct proportion. For verification of non-deterministic systems, mostly non-deterministic Büchi automata (NBA) are used [Cou99, DGV99, EH00, SB00, GO01, GL02, Fri03, BKŘS12, DLLF$^+$16] since they are typically very small and easy to produce.

In contrast to verification of non-deterministic systems, verification of probabilistic systems, such as Markov decision processes (MDP), or synthesis require either more involved techniques, e.g. [KPV06], restrictions to logical fragments, e.g. [AT04, BJP$^+$12], or other types of automata than NBA as detailed below.

Probabilistic LTL model checking cannot profit directly from NBA. Even the qualitative question, whether a formula holds with probability 0 or 1, requires automata with at least a restricted form of determinism. The prime example are the limit-deterministic (also called semi-deterministic) Büchi

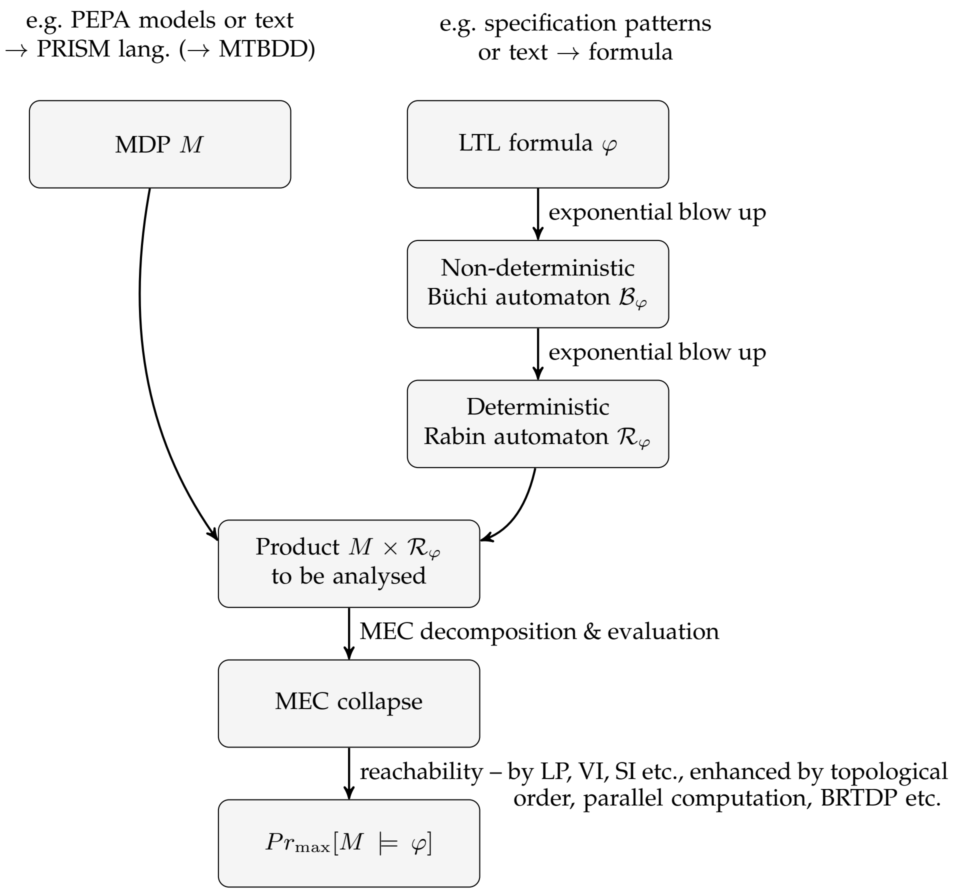

e.g. PEPA models or text
→ PRISM lang. (→ MTBDD)
e.g. specification patterns
or text → formula
MDP M
LTL formula φ
exponential blow up
Non-deterministic
Büchi automaton $\mathcal{B}_\varphi$
exponential blow up
Deterministic
Rabin automaton $\mathcal{R}_\varphi$
Product $M \times \mathcal{R}_\varphi$
to be analysed
MEC decomposition & evaluation
MEC collapse
reachability – by LP, VI, SI etc., enhanced by topological
order, parallel computation, BRTDP etc.
$Pr_{\max}[M \models \varphi]$

Figure 3.1: Traditional probabilistic LTL model checking for MDP, as implemented in e.g. PRISM

automata (LDBA) [CY88]. However, for the general quantitative questions, where the probability of satisfaction is computed, general limit-determinism is not sufficient. Instead, deterministic Rabin automata (DRA) have been mostly used [KNP02], see Figure 3.1. In principle, all standard types of deterministic automata are applicable here except for deterministic Büchi automata (DBA), which are not as expressive as LTL. However, other types of automata, such as deterministic Muller and deterministic parity automata (DPA) are typically larger than DGRA in terms of acceptance condition or the state space, respectively. Indeed, note that every DGRA can be written as a Muller automaton on the same state space with an exponentially-sized acceptance condition, and DPA are a special case of DRA and thus DGRA.

Recently, specific LDBA were also proved applicable to the quantitative setting [HLS$^+$15].

LTL synthesis can also be solved using the automata-theoretic approach. Although DRA and DGRA transformed into games can be used here, the algorithms for the resulting Rabin games [PP06] are not very efficient in practice. In contrast, DPA may be larger, but in this setting they are the automata of choice due to the good practical performance of parity-game solvers [FL09, ML16, JBB$^+$17, MSL18].

Types of translations. The translations of LTL to NBA, e.g., [VW86], are typically *"semantic"* in the sense that each state is given by a set of logical formulae and the language of the state can be captured in terms of semantics of these formulae. In contrast, the determinization of Safra [Saf88] or its improvements [Pit06, Sch09, TD14, FL15] are not "semantic" in the sense that they ignore the structure and produce trees as the new states that, however, lack the logical interpretation. As a result, if we apply Safra's determinization on semantically created NBA, we obtain DRA that lack the structure and, moreover, are unnecessarily large since the construction cannot utilize the original structure. In contrast, our previous work [KE12, KLG13, EK14] as well as some follow-ups [KV15, KV17] provide "semantic" constructions, often producing smaller automata.

3.2 Contributions

We provide direct semantic translations of LTL into deterministic automata several semanticity-preservning transformation. The zoo of the main translations is depicted in Figure 3.2.

In [ATVA'14b] we implement our previous construction of D(G)RA [KE12] and in [FMSD'16] we provide its logic-based presentation, a mechanized Isabelle proof and fix a previous bug. In [CAV'16], we simplify the construction and obtain LDBA of a particular kind that we prove applicable to quantitative LTL model checking (in contrast to the general LDBA). In [ATVA'16] we implement both the construction as well as the model checking procedure. In [TACAS'17a, TACAS'17b] we provide two very different approaches to obtain DPA, applicable to LTL synthesis. The former transforms our LDBA, whereas the latter is a more efficient variation on the classical index appearance record. Both preserve the semantic description, allowing for further optimizations of the resulting automata. In [LICS'18a], we finally provide an asymptotically optimal and unified translation of LTL into D(G)RA, LDBA and NBA, which is additionally simpler than the pre-

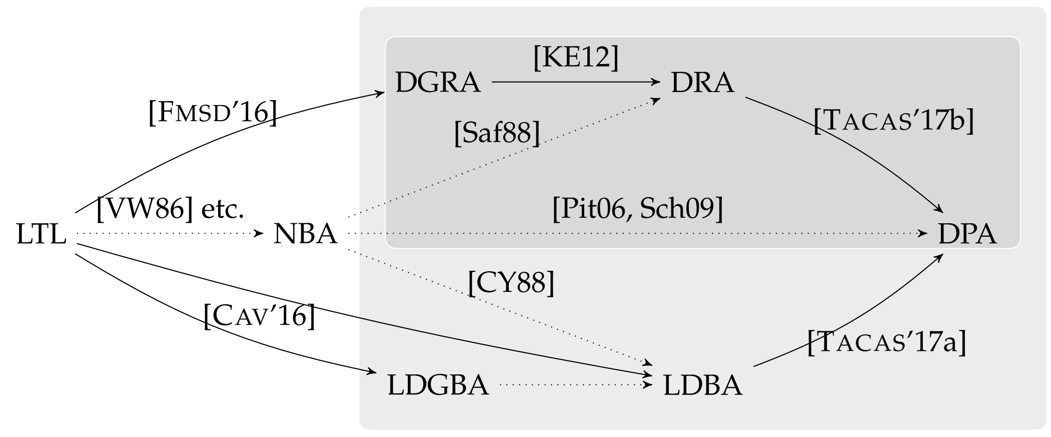


Figure 3.2: Translations of LTL into different types of automata. Automata names are strings of the form (N | D | LD)G?(B | R | P)A, where the meaning of the symbols is non-deterministic, deterministic, limit-deterministic, generalized, Büchi, Rabin, parity, and automaton, respectively.
Translations implemented in `Rabinizer 4` [CAV'18b] are indicated with a solid line. The traditional approaches are depicted as dotted arrows. The determinization of NBA to DRA is implemented in `ltl2dstar` [Kle], to LDBA in `Seminator` [BDK$^+$17] and to (mostly) DPA in `spot` [DLLF$^+$16]. The light gray area denotes the types of automata applicable to probabilistic LTL model checking, while the dark gray denotes applicability to LTL synthesis.

vious translations and more systematic.

Moreover, we provide mature tool support for all these operations and more. While `Rabinizer 3` [ATVA'14b] implements only the translations to DGRA and DRA, `Rabinizer 4` [CAV'18a] implements all the translations depicted in Figure 3.2 with solid arrows. It improves all these translations, both algorithmically and implementation-wise, and moreover, features the first implementation of the translation of a frequency extension of LTL, for further details see Chapter 4. The tool outputs the automata in the Hanoi omega-automata (HOA) format, which we established in [CAV'15b].

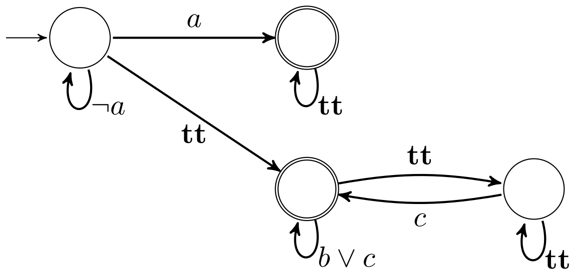
Further, in order to utilize the resulting automata for verification, `Rabinizer 4` comes with our own distribution$^{(*)}$ of the PRISM model checker [KNP02], which allows for model checking MDP against LTL using not only DRA and

(∗). Our distribution additionally features optimized data structures and algorithms such as the BRTDP family, discussed in Chapter 2.

DGRA, but also using LDBA and against frequency LTL using so-called DGRMA [Lpar'15], see Chapter 4. Finally, the tool can turn the produced DPA into parity games between the players with input and output variables. Therefore, when linked to parity-game solvers, it can be also used for LTL synthesis. `Rabinizer 4` is freely available at `http://rabinizer.model.in.tum.de` together with an on-line demo, visualization, usage instructions and examples.

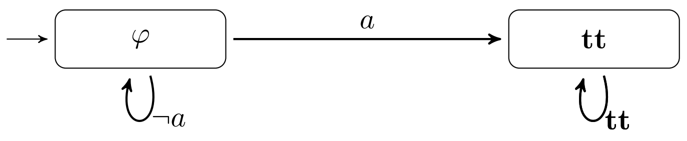
Finally, the infrastructure of `Rabinizer` has been modularized and made easily re-usable as a library `Owl` [Atva'18a]. It has already demonstrated its re-usability in several projects, also without the presence of the library authors. For instance, our experience with Master students has demonstrated that a tool for a complex translation, such as [BDK$^+$17], can be easily implemented using roughly 400 lines of code, achieving performance comparable to the original dedicated tool. We have also implemented Safra's determinization procedure from NBA to DPA. Although this procedure is often described as tedious to implement, it required only 60 lines of code in Owl for the algorithms and 60 lines of code for simple data structures and integration into the pipeline of the tool.

Example. Consider the formula $\varphi = \mathbf{F}a \vee \mathbf{FG}(b \vee \mathbf{F}c)$. The classical approach to construct a non-deterministic automaton would yield the NBA below, which non-deterministically decides whether to wait for an a or not:

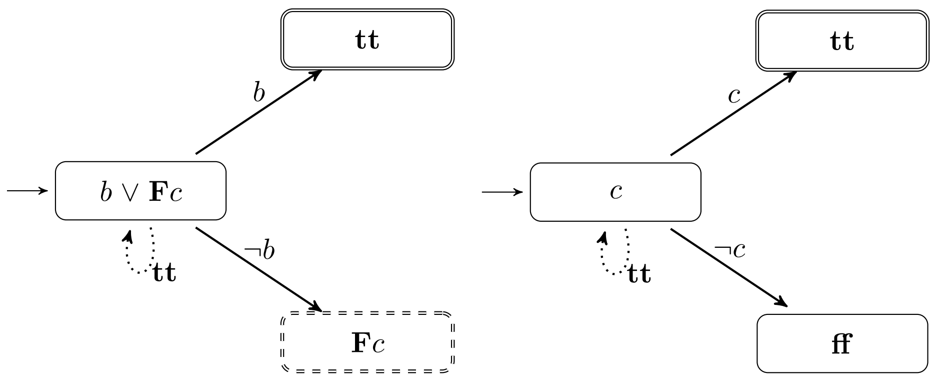


Safra's determinization transforms it into a deterministic automaton with several dozens of states. The state-of-the-art tools `ltl2dstar` [Kle] with `ltl3ba` [BKŘS12] or `spot` [DLLF$^+$16], employing sophisticated simplifications, yield an automaton of around five states. In contrast, `Rabinizer` yields an automaton with two states (the next one below) with a (transition-based) Rabin acceptance condition, not using any simplifications.

The approaches of [KE12, Lics'18a] and [EK14, Fmsd'16] perform this construction as the product of the following automata. First, the master automaton monitors the satisfaction of $\mathbf{F}a$ only:

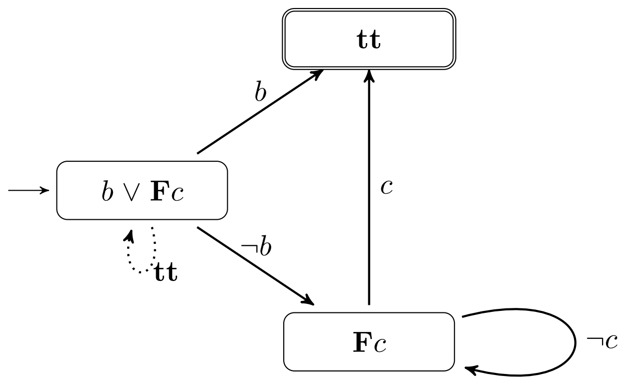


Secondly, the second disjunct $\mathbf{FG}(b \vee \mathbf{F}c)$ is a prefix-independent property and thus is monitored by a set of slave automata. In the former case of [KE12, LICS'18a], this is an automaton for $\mathbf{G}(b \vee \mathbf{F}c)$ (on the left), which relays monitoring $\mathbf{F}c$ to another automaton (on the right):



These slave automata have two special features. Firstly, both automata have a "universal branching" in the initial state, meaning a "copy" of each automaton is started in each step and we monitor whether we accept with all but finitely many copies (the left automaton for a **G**-formula) and with infinitely many copies (the right automaton for an **F**-formula), respectively Secondly, whether the state $\mathbf{F}c$ (of the left slave) is accepting or not depends on the runtime information provided by the slave on the right.

In the case of [EK14, FMSD'16], there is only one slave automaton for the whole $\mathbf{G}(b \vee \mathbf{F}c)$ in which $\mathbf{F}c$ is treated directly:



△

3.3 Contributed papers and activities

[ATVA'14b] Zuzana Komárková and Jan Křetínský. Rabinizer 3: Safraless translation of LTL to small deterministic automata. In *ATVA*, volume 8837 of *LNCS*, pages 235–241. Springer, 2014.

[CAV'15b] Tomáš Babiak, Frantisek Blahoudek, Alexandre Duret-Lutz, Joachim Klein, Jan Křetínský, David Müller, David Parker, and Jan Strejček. The Hanoi omega-automata format. In *CAV (1)*, volume 9206 of *LNCS*, pages 479–486. Springer, 2015.

[CAV'16] Salomon Sickert, Javier Esparza, Stefan Jaax, and Jan Křetínský. Limit-deterministic Büchi automata for linear temporal logic. In *CAV (2)*, volume 9780 of *LNCS*, pages 312–332. Springer, 2016.

[ATVA'16] Salomon Sickert and Jan Křetínský. Mochiba: Probabilistic LTL model checking using limit-deterministic Büchi automata. In *ATVA*, volume 9938 of *LNCS*, pages 130–137, 2016.

[FMSD'16] Javier Esparza, Jan Křetínský, and Salomon Sickert. From LTL to deterministic automata - A safraless compositional approach. *Formal Methods in System Design*, 49(3):219–271, 2016.
Based on a conference paper, which is a part of one of the author's PhD theses

[TACAS'17a] Javier Esparza, Jan Křetínský, Jean-François Raskin, and Salomon Sickert. From LTL and limit-deterministic Büchi automata to deterministic parity automata. In *TACAS (1)*, volume 10205 of *LNCS*, pages 426–442, 2017.
Attached in Part 2 of the thesis.

[TACAS'17b] Jan Křetínský, Tobias Meggendorfer, Clara Waldmann, and Maximilian Weininger. Index appearance record for transforming rabin automata into parity automata. In *TACAS (1)*, volume 10205 of *LNCS*, pages 443–460, 2017.

[LICS'18a] Javier Esparza, Jan Křetínský, and Salomon Sickert. One theorem to rule them all: A unified translation of LTL into ω-automata. In *LICS*, pages 384–393. ACM, 2018.
Attached in Part 2 of the thesis.

[Cav'18a] Jan Křetínský, Tobias Meggendorfer, Salomon Sickert, and Christopher Ziegler. Rabinizer 4: From LTL to your favourite deterministic automaton. In *CAV (1)*, volume 10981 of *LNCS*, pages 567–577. Springer, 2018.
Attached in Part 2 of the thesis.

[Atva'18a] Jan Křetínský, Tobias Meggendorfer, and Salomon Sickert. Owl: A library for ω-words, automata, and LTL. In *ATVA*, volume 11138 of *LNCS*, pages 543–550. Springer, 2018.

Activities The paper [Fmsd'16] was originally invited to the Journal of ACM and accepted for publication there. After acceptance, the authors discovered a bug, withdrew the paper and fixed the bug later.

Merging the features of our PRISM distribution into the public release of PRISM as well as linking the new version of Rabinizer is subject to current collaboration with the authors of PRISM. So far, PRISM newly supports only model checking using DGRA and linking to `Rabinizer 3` [Atva'14b] or any tool producing D(G)RA in the HOA format [Cav'15b]. `Strix` [MSL18] is a tool for LTL synthesis that uses SI to solve games produced by `Rabinizer 4`. The tool won all categories of the LTL/TLSF track at SyntComp 2018[(†)]. This shows that traditional explicit synthesis can be competitive once the sizes of automata drop by orders of magnitude compared to those produced by Safra-like constructions. This is possible due to the fundamentally different approach of `Rabinizer 4`.

The author has given an invited talk at Highlights'18 on the topic of [Lics'18a]. In 2016, the author has obtained a German Research Foundation grant *Verified Model Checkers* on this topic.

(†). For the results of the competition see `http://www.syntcomp.org/syntcomp-2018-results/`

Chapter 4

From Reachability and Expectation to Complex and Group-by Objectives

In Chapter 2 we have dealt with basic objectives such as reachability probability or expected long-run average reward. In Chapter 3 we have shown how the automata-theoretic methods can be used to extend the results to linear temporal logic by reduction to reachability. Each of the objectives is defined by a payoff function on runs of the system, e.g. long-run average reward, and then the results for each of the runs are combined into a single number using an “aggregate” operator, in the previous cases by the expectation operator. In this chapter, we discuss (1) more complex payoff functions and (2) more complex aggregate operators.

4.1 State of the art

4.1.1 Logical and behavioural specifications

There are two fundamentally different approaches to specifying and verifying properties of systems. Firstly, the *logical* approach makes use of specifications given as formulae of temporal or modal logics. Secondly, the behavioural approach exploits various equivalence or refinement checking methods, provided the specifications are given in the same formalism as implementations.

Probabilistic CTL Temporal logics are a convenient and useful formalism to describe behaviour of dynamical systems. In Chapter 3, we considered a probabilistic interpretation of LTL, which allows us to quantify the probability of runs satisfying a given LTL formula. Similarly, probabilistic CTL (PCTL) [HS86, HJ94] is the probabilistic extension of the branching-time logic CTL [EH82], obtained by replacing the existential and universal path quantifiers with the probabilistic operators, which allow us to quantify the probability of runs satisfying a given path formula. At first, the probabilities used were only 0 and 1 [HS86], giving rise to the *qualitative PCTL (qPCTL)*. This has been extended to any values from [0, 1] in [HJ94], yielding the *(quantitative) PCTL* (PCTL). A simple example of a PCTL formula is $ok\mathbf{U}^{=1}(\mathbf{X}^{\geq 0.9}\mathit{finish})$, which says that on almost all runs we reach a state where there is 90% chance to finish in the next step and up to this state ok holds true. Like with probabilistic LTL, PCTL formulae are interpreted over Markov chains where each state is assigned a subset of atomic propositions that are valid in a given state.

The PCTL model checking problem problem has been studied both for finite and infinite Markov chains and decision processes, see e.g. [CY95, HK97, EY09, EKM06, BKS05]. Beside the model checking problem, it is interesting to study the *satisfiability problem*, asking whether a given formula has a *model*, i.e. whether there is a Markov chain satisfying it. If a model does exist, we also want to construct it. Apart from being a fundamental problem, it is a possible tool for checking consistency of specifications or for reactive synthesis. Indeed, the underspecified system together with the specification can be encoded in a formula; the model of such a formula yields a controller for the original system that satisfies the specification. The problem has been shown EXPTIME-complete for qualitative PCTL in the setting where we quantify over finite models (*finite satisfiability*) [HS86, BFKK08] as well as over generally countable models (*infinite satisfiability*) in our [BFKK08]. The problem for (the general quantitative) PCTL remains open for decades.

The satisfiability problem for qPCTL and qPCTL* was investigated already in the early 80's [LS83, KL83, HS86], together with the existence of sound and complete axiomatic systems. The decidability for qPCTL over countable models also follows from these general results for qPCTL*, but the complexity was not examined until [BFKK08], showing it is also EXPTIME-complete, both for finite and infinite satisfiability.

As for the *non-probabilistic* predecessors of PCTL, the satisfiability problem is known to be EXPTIME-complete for CTL [EH82], the same holds for the more general modal μ-calculus [BB87, FL79]. The complexity of the satisfiability problems has been investigated also for fragments of CTL [KV00] and the modal μ-calculus [HKM06].

The PCTL *strategy synthesis* problem asks whether the non-determinism in a given Markov decision process can be resolved so that the resulting Markov chain satisfies the formula [BGL$^+$04, KS08, BBFK06].

Frequency LTL Many properties specifying the desired behaviour, such as "the system is always responsive" can be easily captured by LTL. This logic is in its nature qualitative and cannot express *quantitative* linear-time properties such as "a given failure happens only *rarely*". To overcome this limitation, especially apparent for stochastic systems, extensions of LTL with *frequency* operators have been recently studied [BDL12a, BMM14]. Such extensions come at a cost, and for example the "frequency until" operator can make the controller-synthesis problem undecidable already for non-stochastic systems [BDL12a, BMM14]. It turns out [FK15, THY11, THHY12] that a way of providing significant added expressive power while preserving tractability is to extend LTL only by the "frequency globally" formulae $\mathbf{G}^{\geq f}\varphi$. Such a formula is satisfied if the long-run frequency of satisfying φ on an infinite path is at least f. The respective logic is called *frequency LTL (fLTL)*. MDP controller synthesis for fLTL has been shown decidable for the fragment containing only the operator $\mathbf{G}^{\geq 1}$ [FK15].

Frequency LTL was studied in another variant in [BDL12a, BMM14] where a *frequency until* operator is introduced in two different LTL-like logics, and undecidability is proved for related problems. The work [BDL12a] also yields decidability with restricted nesting of the frequency until operator; as the decidable fragment in [BDL12a] does not contain frequency-globally operator, it is not possible to express many useful properties expressible in our logic. A logic that speaks about frequencies on a finite interval was introduced in [THY11], but the paper provides algorithms only for Markov chains and a bounded fragment of the logic.

As we see in Section 4.2.1, this is related to combining LTL and the mean-payoff objective. There are several works that combine mean-payoff objectives with e.g. logics or parity objectives, but in most cases only simple atomic propositions can be used to define the payoff [BCHJ09, BCHK11, CD11]. The work [BKKW14] extends LTL with another form of quantitative operators, allowing accumulated weight constraint expressed using automata, again not allowing quantification over complex formulas. Further, [ABK14] introduces a variant of LTL with a discounted-future operator.

Linear Distances The distance between processes s and t is typically formalized as $\sup_{p\in\mathcal{C}} |p(s) - p(t)|$ where $\mathcal{C}$ is a class of properties of interest and $p(s)$ is a quantitative value of the property p in process s [DGJP99].

This notion has been introduced in [DGJP99] for Markov chains and further developed in various settings, such as Markov decision processes [FPP04], quantitative transition systems [dAMRS07], or concurrent games [dAFS04].

Several kinds of distances have been investigated for Markov chains. On the one hand, [Aba13, DGJP99, vBW06, vBSW07, BBLM13c, BBLM13b, BBLM13a, GP11], lift the equivalence given by the probabilistic bisimulation of Larsen and Skou [LS89] into *branching distances*. On the other hand, there are *linear distances*. They are particularly appropriate when (i) we are interested in linear-time properties, and/or (ii) we want to estimate the distance based only on simulation runs of the system, i.e. in a black-box setting. (Recall that for branching distances, the underlying probabilistic bisimulation corresponds to testing equivalence where not only runs from the initial state can be observed, but also the current state of the system can be dumped at any moment and system copies restarted from that state [LS89].

There are two main linear distances traditionally considered for Markov chains: total variation distance and trace distance. Algorithms have been proposed for both of them in the case when the Markov chains are known (white-box setting).

Firstly, for the *total variation distance* in the white-box setting, [CK14] shows that deciding whether it is 1 can be done in polynomial time, but computing it is NP-hard and not known to be decidable, however, it can be approximated; [BBLM15b] considers this distance more generally for semi-Markov processes, provides a different approximation algorithm, and shows it coincides with distances based on (i) metric temporal logic, and (ii) timed automata languages.

Secondly, *trace distance* is based on the notion of trace equivalence, which can be decided in polynomial time [DHR08] (however, trace refinement on Markov decision processes is already undecidable [FKS16]). Variants of trace distance are considered in [JMLM14] where it is taken as a limit of finite-trace distances, possibly using discounting or averaging. In [BBLM15a] the finite-trace distance is shown to coincide with distances based on (i) LTL and (ii) LTL without U-operator, i.e., only using X-operator and Boolean connectives; it is also shown NP-hard and not known to be decidable, similarly to the total variation distance; finally, an approximation algorithm is shown (again in the white-box setting), where the over-approximants are branching-time distances, showing a nice connection between the branching and linear distances.

4.1.2 Group-by aggregate operators

The fundamental problem for MDP is to design a strategy resolving the non-deterministic choices so that the systems' behaviour is optimized with respect to a given objective function, or, in the case of multi-objective optimization, to obtain the desired trade-off. The classic objective function (in the optimization phrasing) or the query (in the decision-problem phrasing) consists of two parts. First, a payoff is a measurable function assigning an outcome to each run of the system. It can be real-valued, such as the *long-run average reward* (also called *mean payoff*), or a two-valued predicate, such as *reachability*. Second, the payoffs for single runs are combined into an overall outcome of the strategy, typically in terms of *expectation*. The resulting objective function is then for instance the expected long-run average reward, or the probability to reach a given target state.

In this chapter, firstly, we discuss different *ways how to aggregate the results* on single runs than a single expectation. The motivation to do so is mainly to have a more fine-grained control over the resulting performance, in particular with respect to infrequent ("tail") behaviour and/or more aspects (multiple objectives) at once. Secondly, we discuss how to connect the two parts yet tighter in order to control expectation and other aggregates *at each time point*. The motivation to do so is most apparent in population systems such as many biological systems. We try to unify the philosophy of the two extensions as a "group-by" objective in the contributions section.

Risk-averse control aims to overcome one of the main disadvantages of the expectation operator, namely its ignorance towards the incurred risks, intuitively phrased as a question *"How bad are the bad cases?"* While the standard deviation (or variance) quantifies the spread of the distribution, it does not focus on the bad cases and thus fails to capture the risk. There are a number of quantities used to deal with this issue:

- The *worst-case* analysis (in the financial context known as discounted maximum loss) looks at the payoff of the worst possible run. While this makes sense in a fully non-deterministic environment and lies at the heart of verification, in the probabilistic setting it is typically unreasonably pessimistic, taking into account events happening with probability 0, e.g., never tossing head on a fair coin.

  Risk-averse approaches optimizing the worst case together with expectation have been considered in beyond-worst-case and beyond-almost-sure analysis investigated in both the single-dimensional [BFRR17] and in the multi-dimensional [CR15] setup.

- The *value-at-risk* (VaR) describes the value in the worst p-quantile for some $p \in [0, 1]$. For instance, the value at the 0.5-quantile is the median, for the 0.05-quantile (the ventile) it is the value of the best run among the 5% worst ones. See Figure 4.1 for an example of VaR for two given probability density functions. As such it captures the "reasonably possible" worst-case. There has been an extensive effort spent recently on the analysis of MDP with respect to VaR and the re-formulated notions of quantiles, percentiles, thresholds, satisfaction view etc., see below. Although VaR is more realistic, it tends to ignore the outliers too much, as seen in Figure 4.1 on the right. VaR has been characterized as *"seductive, but dangerous"* and *"not sufficient to control risk"* [Bed95].

  The decision problem related to VaR has been phrased in probabilistic verification mostly in the form *"Is the probability that the payoff is higher than a given value threshold more than a given probability threshold?"*, the so-called *satisfaction semantics* [BBC$^+$14] as opposed to the *expectation semantics*. The total reward gained attention both in the verification community [UB13, HK15, BKKW17] and recently in the AI community [GWX17, LZB17]. Multi-dimensional percentile queries are considered for various objectives, such as mean-payoff, limsup, liminf, shortest path in [RRS17]; for the specifics of two-dimensional case and their interplay, see [BDD$^+$14]; for reachability and LTL, see [EKVY08]. Percentile queries for more complex constraints have also been considered, namely their conjunctions [FKR95, BBC$^+$14] or generally Boolean expressions [HKL17]. Some of these approaches have already been practically applied and found useful by domain experts [BDK$^+$14b, BDK14a].

- The *conditional value-at-risk* (CVaR a.k.a. average value-at-risk, expected shortfall, expected tail loss) answers the question *"What can I expect if things go wrong?"* It is defined as the expectation over the whole worst p-quantile, see Figure 4.1. As such it describes the lossy tail, taking outliers into account, but with the respective weight. In the degenerate cases, CVaR for $p = 1$ is the expectation and for $p = 0$ the worst case. It is an established risk metric in finance, optimization and operations research, e.g. [ADEH99, RU00], and *"is considered to be a more consistent measure of risk than VaR"* [RU00]. Recently, it started permeating to areas closer to verification, such as robotics [CCP16].

There is a body of work that optimizes CVaR in MDP. However, to the best of our knowledge, all the approaches (1) focus on the single-dimensional case, (2) disregard the expectation, and (3) treat neither reachability nor long-run average reward. They focus on the discounted [BO11], total [CCP16], or immediate [KFKT11] reward, as well as extend the results to continuous-time models [HG16, MY17]. This work comes from the area of optimization and operations research, with the notable exception of [CCP16], which focuses on the total reward, generalizing weighted reachability [CCP16]. However, it provides only an approximation solution for the one-dimensional case, neglecting expectation and the respective trade-offs. Further, CVaR is a topic of high interest in finance, e.g., [RU00, Bed95]. The central difference is that they consider variations of portfolios (i.e., the objective functions) while leaving the underlying random process (the market) unchanged. This is dual to our problem, since we fix the objective function and now search for an optimal random process (or the respective strategy).

Distribution transformers Recently, there has been interest in regarding probabilistic systems as deterministic *transformers of probability distributions* rather than individual stochastic processes. In the standard semantics of probabilistic systems, when a probabilistic step from a state to a distribution is taken, the random choice is resolved and we continue from one of the successor states. In contrast, under the *distribution semantics* the choice is not resolved and we continue from the distribution over the successors. Thus, instead of the current state the transition changes the current distribution over the states. This semantics is adequate in many applications, such as systems biology, sensor networks, robot planning, etc. [Hen12, BBMR08, HMW09].

In [KVAK10] model checking MDP under this semantics is shown undecidable and a decidable subclass identified. Approximative approaches to model checking MC are considered in [AAGT15]. A simpler problem of synchronization in MDP under the distribution semantics is solved in [DMS14a, DMS14b].

This semantics has been reflected in the study of bisimulations. The theory of bisimulations is a well-established and elegant framework to describe equivalence between processes based on their behaviour. The original definition was given for non-deterministic processes [Par81] and was further extended to finite probabilistic systems in [LS89]. Since then many variants of bisimulations have been proposed and investigated, in particular an extension to non-deterministic probabilistic systems [SL94]. The distribution semantics has been considered for finite MC [DHR08], finite MDP [FZ14], and finite Markov automata (continuous-time systems) [SZG12]. Further, [CR11, Hen12, Cat05] consider standard bisimulations lifted to distributions, coinciding with the standard bisimulation when projected to Dirac distributions. Similarly, [EHZ10, EHK^{+}13, DH13] consider lifting that, however, differs from the state-based bisimulation in the weak case.

4.2 Contributions

4.2.1 Logical and behavioural specifications

Probabilistic CTL In [Concur'18b], we address the satisfiability problem on fragments of PCTL. In order to get a better understanding of this ultimate problem, we answer the problem for several fragments of PCTL that are

- quantitative, i.e. involving also probabilistic quantification over arbitrary rational numbers (not just 0 and 1),
- step unbounded, i.e. not imposing any horizon for the temporal operators.

Besides, we consider models with unbounded size, i.e. countable models or finite models, but with no a priori restriction on the size of the state space. These are the three distinguishing features, compared to other works. Firstly, solutions for the *qualitative* PCTL have been given in [HS86, BFKK08] and for a more general qualitative logic PCTL* in [LS83, KL83]. Secondly, [CK16] shows decidability for *bounded PCTL* where the scope of the operators is restricted by a step bound to a given time horizon. Thirdly, the *bounded satisfiability problem* is to determine, whether there exists a model of a given size for a given formula. This problem has been solved by encoding it into an SMT problem [BFS12].

In particular, we show decidability of the satisfiability problem for several quantitative unbounded fragments of PCTL, focusing on future- and globally-operators (**F**,**G**), and discuss both finite and infinite satisfiability. Further, we identify a "smallest elegant" fragment where the problem remains open and the solution requires additional techniques.

Frequency LTL In [LPAR'15] we make a step towards the ultimate goal of a model checking procedure for the whole fLTL. We address the general *quantitative* setting with arbitrary frequency bounds p and consider the fragment $\mathrm{LTL}_{\setminus\mathbf{GU}}$, which is obtained from frequency LTL by preventing the $\mathbf{U}$ operator from occurring inside $\mathbf{G}$ or $\mathbf{G}^{\geq f}$ formulas (but still allowing the $\mathbf{F}$ operator to occur anywhere in the formula). The approach we take is completely different from [FK15] where ad hoc product MDP construction is used, heavily relying on existence of certain types of strategies in the $f = 1$ case. In this paper we provide, to the best of our knowledge, the first translation of a quantitative logic to equivalent *deterministic* automata. This allows us to take the standard automata-theoretic approach to verification [VW86]: after obtaining the finite automaton, we do not deal with the structure of the formula originally given, and we solve a synthesis problem on a product of the single automaton with the MDP.

To our best knowledge, this paper gives the first decidability result for probabilistic verification against linear-time temporal logics extended by *quantitative* frequency operators with *complex nested subformulas* of the logic.

It works in two steps, keeping the same time complexity as for ordinary LTL. In the first step, a $\mathrm{LTL}_{\setminus\mathbf{GU}}$ formula gets translated to an equivalent *deterministic* generalized Rabin automaton extended with mean-payoff objectives. This step is inspired by our previous work [KLG13], but the extension with auxiliary automata for $\mathbf{G}^{\geq f}$ requires a different construction. The second step is the analysis of MDPs against conjunction of limit inferior mean-payoff, limit superior mean-payoff, and generalized Rabin objectives. This result is obtained by adapting and combining several existing involved proof techniques of [BCFK13] and our [LICS'15]..

Although our algorithm does not allow us to handle the extension of the whole LTL, the considered fragment $\mathrm{LTL}_{\setminus\mathbf{GU}}$ contains a large class of formulas and offers significant expressive power. It subsumes the GR(1) fragment of LTL [BJP$^+$12], which has found use in synthesis for hardware designs. The $\mathbf{U}$ operator, although not allowed within a scope of a $\mathbf{G}$ operator, can still be used for example to distinguish paths based on their prefixes.

Example. As an example synthesis problem expressible in this fragment, consider a cluster of servers where each server plays either a role of a load-balancer or a worker. On startup, each server $\underline{l}$istens for a message specifying its role. A load-$\underline{b}$alancer $\underline{f}$orwards each $\underline{r}$equest and only waits for a $\underline{c}$onfirmation whereas a $\underline{w}$orker $\underline{p}$rocesses the requests itself. A specification for a single server in the cluster can require, for example, that the following formula (with propositions explained above) holds with probability at least 0.95:

$$\Big(\big(l\,\mathbf{U}\,b\big) \to \mathbf{G}^{\geq 0.99}\big(r \to \mathbf{X}(f \wedge \mathbf{F}c)\big)\Big) \wedge \Big(\big(l\,\mathbf{U}\,w\big) \to \mathbf{G}^{\geq 0.85}\big(r \to (\mathbf{X}p \vee \mathbf{XX}p)\big)\Big)$$

$\triangle$

Previous work [THHY12] offered only translation of a similar logic to *non-deterministic* "mean-payoff Büchi automata" noting that it is difficult to give an analogous reduction to *deterministic* "mean-payoff Rabin automata". The reason is that the non-determinism is inherently present in the form of guessing whether the subformulas of $\mathbf{G}^{\geq f}$ are satisfied on a suffix. Our construction overcomes this difficulty and offers equivalent deterministic automata, utilizing our [KE12, KLG13, LICS'15].

Linear distances In [CONCUR'16], we introduce a simple framework for linear distances between Markov chains by the formula $\sup_{p \in \mathcal{C}} |p(s) - p(t)|$ above. Here $p(s)$ is the probability of satisfying p when starting a simulation run in state s. In other words, when p is seen as a language it is the probability to generate a trace belonging to p.

We consider estimating distances only from simulating the systems, i.e. the *black-box setting*. One of the main difficulties is that the class $\mathcal{C}$ typically includes properties with arbitrarily long horizon or even infinite-horizon properties, whereas every simulation run is necessarily finite. Note that we do not want employ here any simplifications such as imposed fixed horizon or discounting, typically used for obtaining efficient algorithms, e.g., [DGJP99, vBW06, BBLM13b], and the undiscounted setting is fundamentally more complex [vBSW07]. Since even simpler tasks are impossible for unbounded horizon in the black-box setting without any further knowledge, it is assumed we know an upper bound on the size of the state space $|S|$ and a lower bound on the minimum transition probability p_{min}, similarly to Section 2.2.2.

Depending on the class $\mathcal{C}$, we obtain various interesting instantiations of our framework:

Example. One extreme choice is to consider all measurable languages, resulting in the *total variation distance*. The other extreme choices are to consider (1) only the generators of the σ-algebra, i.e. the cones, resulting in the *finite-trace distance*; or (2) only the elementary events, resulting in the *infinite-trace distance*. There are many possible choices for $\mathcal{L}$ between the two extremes above, such as classes of the Borel hierarchy, long-run average reward criteria, ω-regular languages, classes of automata or sets of temporal formulae of certain form (the class of ω-regular languages can also be given by the monadic second-order logic or ω-automata) etc. $\triangle$

We can now use statistics on finite simulation runs to (i) deduce information on the whole infinite runs and (ii) estimate the distance of the systems. For a particular distance function $\mathsf{D}_{\mathcal{L}}$, the goal is to construct an algorithm with the following specification:

Specification of $\mathcal{L}$-distance estimation

Input:

- two finite black-box MCs $\mathcal{M}_1, \mathcal{M}_2$ (i.e., access to any desired finite number of sampled simulation paths of any desired finite lengths)
- confidence $\alpha \in (0,1)$
- interval width $\delta \in (0,1)$

Output: interval I such that $|I| \leq \delta$ and $\mathbb{P}[\mathsf{D}_{\mathcal{L}}(\mathcal{M}_1, \mathcal{M}_2) \in I] \geq 1 - \alpha$

In short, we show that the total variation distance cannot be estimated by simulating the systems, and that the finite-trace distance can be estimated. The former result is further exploited to show that the inestimability result holds also already for clopen sets, Rabin automata, and LTL (even without the Until-operator). However, it is also shown that infinite-trace distance and distances for some fragments of LTL are estimable. Moreover, restricting the size of automata also yields estimability. Furthermore, assuming finite precision of transition probabilities, e.g. they are given by at most two decimal digits, even the total variation distance can be estimated, exploiting the white-box algorithms. Under this assumption, *trace equivalence* can also be decided correctly with arbitrarily high probability.

4.2.2 Group-by aggregate operators

Risk-averse control A more focused view on differences among possible behaviours of a system can be taken by aggregating similar behaviours, tailoring an abstract view. We draw an analogy with database queries. Instead of looking at all data points, we can aggregate the data by functions such as AVG, corresponding to the expectation semantics, but also MIN when the worst-case/sure-winning view is taken, or MAX in the analogous situation. Beyond that, we can aggregate in a more sophisticated manner. A query

```
select count(*) from runs
where value = true    //alternatively, value≥threshold
```

corresponds to the satisfaction semantics. A slight variation

```
select value, count(*) from runs
group by value
```

corresponds more closely to the quantile analysis and CVaR analysis. One can even think of the distribution semantics as grouping according to the state at a particular moment:

```
select state[t], count(*) from runs
group by state[t]
```

In [Lics'15], we unified the expectation semantics and the satisfaction semantics for long-run average reward as follows. Intuitively, the problem we consider asks to *optimize* the expectation while *ensuring* the satisfaction. Formally, consider an MDP with n reward functions, a probability threshold vector $\mathbf{p}$ (or threshold p for the so-called joint interpretation [BBC+14]), and a reward threshold vector $\mathbf{r}$. We consider the set of *satisfaction* strategies that ensure the satisfaction semantics, i.e., they ensure with probability at least p (or respective probabilitites of $\mathbf{p}$) that runs have long-run average reward value vector at least $\mathbf{r}$. Then the optimization of the expectation is considered with respect to the satisfaction strategies. Note that if $\mathbf{p}$ is $\mathbf{0}$ (assuming non-negative rewards here for simplicity), then the satisfaction strategies is the set of all strategies and we obtain the traditional expectation semantics as a special case. We investigate the questions of algorithmic complexity, strategy complexity (in terms of memory and randomization), and trade-offs in the sense of a Pareto curve.

Example. We list a few examples, illustrating the use of such combinations:

- For simple risk aversion, consider a single reward function modelling investment. Positive reward stands for profit, negative for loss. We aim at maximizing the expected long-run average while guaranteeing that it is non-negative with at least 95%. This is an instance with $n = 1, p = 0.95, r = 0$.
- For more dimensions, consider the example [Put94, Problems 6.1, 8.17]. A vendor assigns to each customer either a low or a high rank.

Further, there is a decision the vendor makes each year either to invest money into sending a catalogue to the customer or not. Depending on the rank and on receiving a catalogue, the customer spends different amounts for vendor's products and the rank can change. The aim is to maximize the expected profit provided the catalogue is almost surely sent with frequency at most f. Further, one can extend this example to only require that the catalogue frequency does not exceeded f with 95% probability, but 5% best customers may still receive catalogues very often.

- A gratis service for downloading is offered as well as a premium one. For each we model the throughput as rewards r_1, r_2. For the gratis service, expected throughput $1Mbps$ is guaranteed as well as 60% connections running on at least $0.8Mbps$. For the premium service, not only have we a higher expectation of $10Mbps$, but also 95% of the connections are guaranteed to run on at least $5Mbps$ and 80% on even $8Mbps$ (satisfaction constraints). In order to keep this guarantee, we may need to temporarily hire resources from a cloud, whose cost is modelled as a reward r_3. While satisfying the guarantee, we want to maximize the expectation of $q_2 \cdot r_2 - q_3 \cdot r_3$ where q_2 is the price per Mb at which the premium service is sold and q_3 is the price at which additional servers can be hired. △

In [LICS'18a], we investigate optimization of MDP with respect to CVaR as well as the respective trade-offs with expectation and VaR. We study it for the first time with the payoff functions of reachability and mean payoff, which are fundamental in verification. Moreover, we cover both the single-dimensional and the multi-dimensional case and combinations of $\mathbb{E}$, VaR and CVaR.

More specifically, we define CVaR for MDP and show the peculiarities of the concept. Then we study again the computational complexity and the strategy complexity for various settings. Note that CVaR features some surprising behaviour, preventing us to trivially adapt the solutions to the expectation and VaR problems. For instance, compared to expectation and VaR, CVaR does not behave linearly when considering stochastic combination of strategies. Further, already conjunctions of CVaR constraints alone induce an NP-hard problem since we can force a strategy to play deterministically.

Distribution transformers In [CONCUR'14], we consider the distribution semantics of MDP with general state spaces and give a *natural* definition of

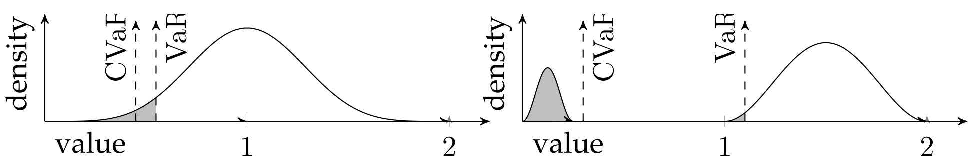


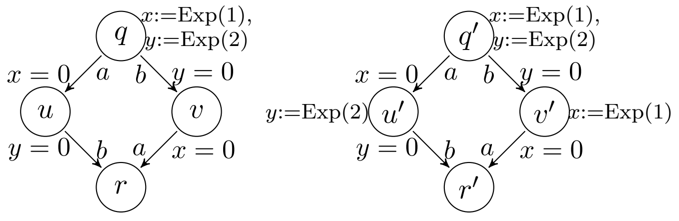
Figure 4.1: Illustration of VaR and CVaR. Given the depicted probability density functions f of a continuous real variable X and a risk probability $p \in [0, 1]$, we have $\mathrm{CVaR}_p(X) = \int_{-\infty}^{\mathrm{VaR}_p(X)} x \cdot f(x)\, dx$.

the respective notions of bisimulation. It arises as an unusual, but very simple instantiation of the standard coalgebraic framework for bisimulations [SR11]. Despite its simplicity, the resulting notion is surprisingly fruitful for two reasons.

Firstly, it is more precise than other equivalences when applied to systems with the distribution semantics such as large-population models where different parts of the population act differently [M$^+$74]. Indeed, as argued in [GA12], some equivalent states are not identified in the standard probabilistic bisimulations and too many are identified in the recent distribution based bisimulations [DHR08, FZ14]. Our approach allows for a bisimulation identifying precisely the desired states [GA12].

Secondly, our bisimulation over distributions induces a novel behavioural equivalence on states. This equivalence is useful, for it identifies states that are behaviourally indistinguishable in many settings, but were unnecessarily distinguished by standard bisimulations. We document this in several applications: ranging from partially observed systems control [SPK13] where the controller only has a probabilistic belief where the system is, over distributed scheduler synthesis [GDF09], to continuous-time systems where random waiting times are sampled, but are not observed before they elapse [DK05]. We illustrate the latter in the following intriguing example from the continuous-time area due to [BDH].

Example. Consider the following two stochastic automata [DK05]. They have “kitchen timers” x and y, which get a random exponentially distributed value when a state with a corresponding assignment is entered. Then the time elapses and the values of the timers decrease until the first one reaches 0. Then a transition to the next state is taken and the other timers keep on running.



If we observe the times when transitions occur (the timer rings), the two systems above behave the same. Indeed, due to the memoryless property of the exponential distributions, there is no way to distinguish them. Yet existing notions of bisimulations and behavioural equivalences in the literature fail to capture this. The distributional view, which we take, is necessary here. Indeed, as u' cannot be matched by any u with a particular remaining time on the timer, but can be matched by an appropriate distribution over u's with all possible remaining times. $\triangle$

Nevertheless, the key idea to work with distributions instead of single states also bears disadvantages. The main difficulty is that even for finite systems the space of distributions is uncountable, thus bisimulation is difficult to compute. However, we show that it admits a concise representation using methods of linear algebra and we give algorithms to compute it. Namely, we give the first algorithm to compute distributional bisimulation on finite non-deterministic probabilistic systems and, further, an algorithm deciding bisimilarity on several classes of uncountable (continuous-time) systems. In order to cover e.g. continuous-time systems, we need to handle both uncountably many states (that store the sampled time) and labels (real time durations). Fortunately, there is an elegant way to do so using the standard coalgebra framework.

4.3 Contributed papers and activities

[CONCUR'14] Holger Hermanns, Jan Krčál, and Jan Křetínský. Probabilistic bisimulation: Naturally on distributions. In *CONCUR*, volume 8704 of *LNCS*, pages 249–265. Springer, 2014.
Attached in Part 2 of the thesis.

[LICS'15] Krishnendu Chatterjee, Zuzana Komárková, and Jan Křetínský. Unifying two views on multiple mean-payoff objectives in Markov decision processes. In *LICS*, pages 244–256. IEEE Computer Society, 2015.
Journal version published in *Logical Methods in Computer*

Science, 13(2), 2017.
Attached in Part 2 of the thesis.

[LPAR'15] Vojtěch Forejt, Jan Krčál, and Jan Křetínský. Controller synthesis for MDPs and frequency $\mathrm{LTL}_{\setminus\mathbf{GU}}$. In *LPAR*, volume 9450 of *LNCS*, pages 162–177. Springer, 2015.

[CONCUR'16] Przemyslaw Daca, Thomas A. Henzinger, Jan Křetínský, and Tatjana Petrov. Linear distances between Markov chains. In *CONCUR*, volume 59 of *LIPIcs*, pages 20:1–20:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

[LICS'18b] Jan Křetínský and Tobias Meggendorfer. Conditional value-at-risk for reachability and mean payoff in Markov decision processes. In *LICS*, pages 609–618. ACM, 2018.
Attached in Part 2 of the thesis.

[CONCUR'18b] Jan Křetínský and Alexej Rotar. The satisfiability problem for unbounded fragments of probabilistic CTL. In *CONCUR*, volume 118 of *LIPIcs*, pages 8:1–8:18. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.

Activities On this topic, the author has submitted a German Research Foundation grant *Group-by Objectives in Probabilistic Verification* in 2018.

This strand of research is the most recent to be systematically investigated. However, the citations of e.g. [CONCUR'14] from areas as diverse as refinement of probabilistic processes, algebras, quantum computation or logic, show the potential for further use and the need for further exploration.

Chapter 5

Conclusion

We have reviewed our work since obtaining the second PhD in mid-2014 in three areas connected to quantitative verification. While the LTL strand has just reached its most important result, the other two strands on utilizing machine learning and revisiting traditional objectives span many options for further work. This also corresponds to the grant proposals in preparation.

Both PhD theses at the Technical University of Munich in 2013 and at Masaryk University in 2014 used previous work of the LTL strand. While the thesis in 2013 is more connected to probabilistic verification, the one in 2014 focuses on modal transition system. This strand has not been entirely left since then, as documented on the relevant publications listed below.

Papers on modal transition systems

[FACS'14] Ulrich Fahrenberg, Jan Křetínský, Axel Legay, and Louis-Marie Traonouez. Compositionality for quantitative specifications. In *FACS*, volume 8997 of *LNCS*, pages 306–324. Springer, 2014. **Journal version published in** *Soft Computing*, 22(4):1139–1158, 2018.

[CBSE'15] Nikola Beneš, Przemyslaw Daca, Thomas A. Henzinger, and Jan Křetínský. Complete composition operators for IOCO-testing theory. In *CBSE*, pages 101–110. ACM, 2015.

[ACTA'15] Nikola Beneš, Jan Křetínský, Kim G. Larsen, Mikael H. Møller, Salomon Sickert, and Jiří Srba. Refinement checking on parametric modal transition systems. *Acta Informatica*, 52(2-3):269–297, 2015.

Based on two conference papers, which are part of one of the author's PhD theses

[BIRTHDAY'17] Jan Křetínský. 30 years of modal transition systems: Survey of extensions and analysis. In *Models, Algorithms, Logics and Tools*, volume 10460 of *LNCS*, pages 36–74. Springer, 2017.
Based on one of the author's PhD theses

Bibliography

[AAGT15] Manindra Agrawal, S. Akshay, Blaise Genest, and P. S. Thiagarajan. Approximate verification of the symbolic dynamics of markov chains. *J. ACM*, 62(1):2:1–2:34, 2015.

[Aba13] Alessandro Abate. Approximation metrics based on probabilistic bisimulations for general state-space Markov processes: A survey. *Electr. Notes Theor. Comput. Sci.*, 297:3–25, 2013.

[ABK14] Shaull Almagor, Udi Boker, and Orna Kupferman. Discounting in LTL. In *TACAS*, 2014.

[ADEH99] Philippe Artzner, Freddy Delbaen, Jean-Marc Eber, and David Heath. Coherent measures of risk. *Mathematical Finance*, 9(3):203–228, 1999.

[AL09] Husain Aljazzar and Stefan Leue. Generation of counterexamples for model checking of markov decision processes. In *QEST*, pages 197–206. IEEE Computer Society, 2009.

[AT04] Rajeev Alur and Salvatore La Torre. Deterministic generators and games for LTL fragments. *ACM Trans. Comput. Log.*, 5(1):1–25, 2004.

[AY17] Gürdal Arslan and Serdar Yüksel. Decentralized q-learning for stochastic teams and games. *IEEE Trans. Automat. Contr.*, 62(4):1545–1558, 2017.

[BB87] Behnam Banieqbal and Howard Barringer. Temporal logic with fixed points. In *Temporal Logic in Specification*, volume 398 of *Lecture Notes in Computer Science*, pages 62–74. Springer, 1987.

[BBB$^+$10] Ananda Basu, Saddek Bensalem, Marius Bozga, Benoît Caillaud, Benoît Delahaye, and Axel Legay. Statistical abstraction and model-checking of large heterogeneous systems. In *FMOODS/FORTE*, pages 32–46, 2010.

[BBB$^+$14] Nathalie Bertrand, Patricia Bouyer, Thomas Brihaye, Quentin Menet, Christel Baier, Marcus Größer, and Marcin Jurdzinski. Stochastic timed automata. *Logical Methods in Computer Science*, 10(4), 2014.

[BBC$^+$08] Jiri Barnat, Lubos Brim, Ivana Cerná, Milan Ceska, and Jana Tumova. ProbDiVinE-MC: Multi-core LTL model checker for probabilistic systems. In *QEST*, pages 77–78, 2008.

[BBC$^+$14] Tomáš Brázdil, Václav Brožek, Krishnendu Chatterjee, Vojtěch Forejt, and Antonín Kučera. Two views on multiple mean-payoff objectives in Markov decision processes. *LMCS*, 10(1), 2014.

[BBFK06] Tomás Brázdil, Václav Brozek, Vojtech Forejt, and Antonín Kucera. Stochastic games with branching-time winning objectives. In *LICS*, pages 349–358. IEEE Computer Society, 2006.

[BBLM13a] Giorgio Bacci, Giovanni Bacci, Kim G. Larsen, and Radu Mardare. The BisimDist library: Efficient computation of bisimilarity distances for Markovian models. In *QEST*, pages 278–281, 2013.

[BBLM13b] Giorgio Bacci, Giovanni Bacci, Kim G. Larsen, and Radu Mardare. Computing behavioral distances, compositionally. In *MFCS*, pages 74–85, 2013.

[BBLM13c] Giorgio Bacci, Giovanni Bacci, Kim G. Larsen, and Radu Mardare. On-the-fly exact computation of bisimilarity distances. In *TACAS*, pages 1–15, 2013.

[BBLM15a] Giorgio Bacci, Giovanni Bacci, Kim G. Larsen, and Radu Mardare. Converging from branching to linear metrics on Markov chains. In *ICTAC*, pages 349–367, 2015.

[BBLM15b] Giorgio Bacci, Giovanni Bacci, Kim G. Larsen, and Radu Mardare. On the total variation distance of semi-Markov chains. In *FoSSaCS*, pages 185–199, 2015.

[BBMR08] Roberto Baldoni, François Bonnet, Alessia Milani, and Michel Raynal. On the solvability of anonymous partial grids exploration by mobile robots. In *OPODIS*, volume 5401 of *LNCS*, pages 428–445. Springer, 2008.

[BBR14] Aaron Bohy, Véronique Bruyère, and Jean-François Raskin. Symblicit algorithms for optimal strategy synthesis in monotonic markov decision processes. In *Workshop on Synthesis, SYNT*, pages 51–67, 2014.

[BBS08] Lucian Busoniu, Robert Babuska, and Bart De Schutter. A comprehensive survey of multiagent reinforcement learning. *IEEE Trans. Systems, Man, and Cybernetics, Part C*, 38(2):156–172, 2008.

[BCC$^+$14] Tomáš Brázdil, Krishnendu Chatterjee, Martin Chmelík, Vojtěch Forejt, Jan Křetínský, Marta Z. Kwiatkowska, David Parker, and Mateusz Ujma. Verification of Markov decision processes using learning algorithms. In *ATVA*, pages 98–114, 2014.

[BCFK13] Tomás Brázdil, Krishnendu Chatterjee, Vojtech Forejt, and Antonín Kucera. Trading performance for stability in Markov decision processes. In *LICS*, pages 331–340, 2013.

[BCHJ09] Roderick Bloem, Krishnendu Chatterjee, Thomas A Henzinger, and Barbara Jobstmann. Better quality in synthesis through quantitative objectives. In *CAV*. Springer, 2009.

[BCHK11] Udi Boker, Krishnendu Chatterjee, Thomas A Henzinger, and Orna Kupferman. Temporal specifications with accumulative values. In *LICS*. IEEE, 2011.

[BCLS13] Benoît Boyer, Kevin Corre, Axel Legay, and Sean Sedwards. PLASMA-lab: A flexible, distributable statistical model checking library. In *QEST*, pages 160–164, 2013.

[BD96] C. Boutilier and R. Dearden. Approximating value trees in structured dynamic programming. In *In Proceedings of the Thirteenth International Conference on Machine Learning*, pages 54–62, 1996.

[BDD+14] Christel Baier, Marcus Daum, Clemens Dubslaff, Joachim Klein, and Sascha Klüppelholz. Energy-utility quantiles. In *NASA Formal Methods*, pages 285–299, 2014.

[BDG95] C. Boutilier, R. Dearden, and M. Goldszmidt. Exploiting structure in policy construction. In *IJCAI-95, pp.1104–1111*, 1995.

[BDH] Christel Baier, Pedro Ruben D'Argenio, and Holger Hermanns. What is the relation between CTMC and TA? Personal communication.

[BDH99] C. Boutilier, T. Dean, and S. Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *JAIR*, 11:1–94, 1999.

[BDK14a] Christel Baier, Clemens Dubslaff, and Sascha Klüppelholz. Trade-off analysis meets probabilistic model checking. In *CSL-LICS*, pages 1:1–1:10, 2014.

[BDK+14b] Christel Baier, Clemens Dubslaff, Sascha Klüppelholz, Marcus Daum, Joachim Klein, Steffen Märcker, and Sascha Wunderlich. Probabilistic model checking and non-standard multi-objective reasoning. In *FASE*, pages 1–16, 2014.

[BDK+17] František Blahoudek, Alexandre Duret-Lutz, Mikuláš Klokočka, Mojmír Křetínský, and Jan Strejček. Seminator: A tool for semi-determinization of omega-automata. In *LPAR*, pages 356–367, 2017.

[BDL+06] Gerd Behrmann, Alexandre David, Kim Guldstrand Larsen, John Håkansson, Paul Pettersson, Wang Yi, and Martijn Hendriks. UPPAAL 4.0. In *QEST*, pages 125–126, 2006.

[BDL12a] Benedikt Bollig, Normann Decker, and Martin Leucker. Frequency linear-time temporal logic. In *TASE*, Beijing, China, July 2012. IEEE.

[BDL+12b] Peter E. Bulychev, Alexandre David, Kim Guldstrand Larsen, Marius Mikucionis, Danny Bøgsted Poulsen, Axel Legay, and Zheng Wang. UPPAAL-SMC: Statistical model checking for priced timed automata. In *QAPL*, 2012.

[Bed95] Tanya Beder. Var: Seductive but dangerous. 51:12–24, 09 1995.

[Bel57] Richard Bellman. A Markovian decision process. *Indiana Univ. Math. J.*, 6:679–684, 1957.

[BFHH11] Jonathan Bogdoll, Luis María Ferrer Fioriti, Arnd Hartmanns, and Holger Hermanns. Partial order methods for statistical model checking and simulation. In *FMOODS/FORTE*, pages 59–74, 2011.

[BFKK08] T. Brázdil, V. Forejt, J. Křetínský, and A. Kučera. The satisfiability problem for probabilistic CTL. In *LICS*, pages 391–402, 2008.

[BFRR17] Véronique Bruyère, Emmanuel Filiot, Mickael Randour, and Jean-François Raskin. Meet your expectations with guarantees: Beyond worst-case synthesis in quantitative games. *Inf. Comput.*, 254:259–295, 2017.

[BFS12] Nathalie Bertrand, John Fearnley, and Sven Schewe. Bounded satisfiability for PCTL. In *CSL*, volume 16 of *LIPIcs*, pages 92–106. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.

[BGC04] Christel Baier, Marcus Größer, and Frank Ciesinski. Partial order reduction for probabilistic systems. In *QEST*, pages 230–239, 2004.

[BGL$^+$04] Christel Baier, Marcus Größer, Martin Leucker, Benedikt Bollig, and Frank Ciesinski. Controller synthesis for probabilistic systems. In *IFIP TCS*, volume 155 of *IFIP*, pages 493–506. Kluwer/Springer, 2004.

[BHH12] Jonathan Bogdoll, Arnd Hartmanns, and Holger Hermanns. Simulation and statistical model checking for modestly nondeterministic models. In *MMB/DFT*, pages 249–252, 2012.

[BJP$^+$12] Roderick Bloem, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. Synthesis of reactive(1) designs. *J. Comput. Syst. Sci.*, 78(3):911–938, 2012.

[BKH99] Christel Baier, Joost-Pieter Katoen, and Holger Hermanns. Approximate symbolic model checking of continuous-time markov chains. In *CONCUR*, pages 146–161, 1999.

[BKKW14] Christel Baier, Joachim Klein, Sascha Klüppelholz, and Sascha Wunderlich. Weight monitoring with linear temporal logic: Complexity and decidability. In *CSL-LICS*. ACM, 2014.

[BKKW17] Christel Baier, Joachim Klein, Sascha Klüppelholz, and Sascha Wunderlich. Maximizing the conditional expected reward for reaching the goal. In *TACAS*, pages 269–285, 2017.

[BKL$^+$17] Christel Baier, Joachim Klein, Linda Leuschner, David Parker, and Sascha Wunderlich. Ensuring the reliability of your model checker: Interval iteration for markov decision processes. In *CAV*, pages 160–180, 2017.

[BKŘS12] Tomáš Babiak, Mojmír Křetínský, Vojtěch Řehák, and Jan Strejček. LTL to Büchi automata translation: Fast and more deterministic. In *TACAS*, pages 95–109, 2012.

[BKS05] Tomás Brázdil, Antonín Kucera, and Oldrich Strazovský. On the decidability of temporal properties of probabilistic pushdown automata. In *STACS*, pages 145–157. Springer, 2005.

[BKW14] Nicolas Basset, Marta Z. Kwiatkowska, and Clemens Wiltsche. Compositional controller synthesis for stochastic games. In *CONCUR*, pages 173–187, 2014.

[BMM14] Patricia Bouyer, Nicolas Markey, and Raj Mohan Matteplackel. Averaging in LTL. In *CONCUR*, volume 8704 of *LNCS*. Springer, 2014.

[BO11] Nicole Bäuerle and Jonathan Ott. Markov decision processes with average-value-at-risk criteria. *Math. Meth. of OR*, 74(3):361–379, 2011.

[BT00] Ronen I. Brafman and Moshe Tennenholtz. A near-optimal polynomial time algorithm for learning in certain classes of stochastic games. *Artif. Intell.*, 121(1-2):31–47, 2000.

[Cat05] Stefano Cattani. *Trace-based process algebras for real-time probabilistic systems*. PhD thesis, University of Birmingham, UK, 2005.

[CCD15] Krishnendu Chatterjee, Martin Chmelik, and Przemyslaw Daca. CEGAR for compositional analysis of qualitative properties in markov decision processes. *Formal Methods in System Design*, 47(2):230–264, 2015.

[CCP16] Stefano Carpin, Yinlam Chow, and Marco Pavone. Risk aversion in finite markov decision processes using total cost criteria and average value at risk. In *ICRA*, pages 335–342, 2016.

[CD11] Krishnendu Chatterjee and Laurent Doyen. Energy and mean-payoff parity Markov decision processes. In *MFCS*. Springer, 2011.

[CDL$^+$10] Benoît Caillaud, Benoît Delahaye, Kim G. Larsen, Axel Legay, Mikkel L. Pedersen, and Andrzej Wasowski. Compositional design methodology with constraint markov chains. In *QEST*, pages 123–132, 2010.

[CFK$^+$13] T. Chen, V. Forejt, M. Kwiatkowska, D. Parker, and A. Simaitis. PRISM-games: A model checker for stochastic multi-player games. In *TACAS*, volume 7795 of *LNCS*, pages 185–191. Springer, 2013.

[CHJS11] Krishnendu Chatterjee, Monika Henzinger, Manas Joglekar, and Nisarg Shah. Symbolic algorithms for qualitative analysis of markov decision processes with büchi objectives. In *CAV*, pages 260–276, 2011.

[CHR10] Pavol Cerný, Thomas A. Henzinger, and Arjun Radhakrishna. Simulation distances. In *CONCUR*, volume 6269 of *Lecture Notes in Computer Science*, pages 253–268. Springer, 2010.

[CK91] D. Chapman and L. P. Kaelbling. Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. pages 726–731. Morgan Kaufmann, 1991.

[CK14] Taolue Chen and Stefan Kiefer. On the total variation distance of labelled Markov chains. In *CSL-LICS*, pages 33:1–33:10, 2014.

[CK16] Souymodip Chakraborty and Joost-Pieter Katoen. On the satisfiability of some simple probabilistic logics. In *LICS*, pages 56–65. ACM, 2016.

[Con90] Anne Condon. On algorithms for simple stochastic games. In *Advances In Computational Complexity Theory*, volume 13 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 51–72. DIMACS/AMS, 1990.

[Cou99] Jean-Michel Couvreur. On-the-fly verification of linear temporal logic. In *FM*, pages 253–271, 1999.

[CR11] S. Crafa and F. Ranzato. A spectrum of behavioral relations over ltss on probability distributions. In *CONCUR*, pages 124–139, 2011.

[CR15] Lorenzo Clemente and Jean-François Raskin. Multidimensional beyond worst-case and almost-sure problems for mean-payoff objectives. In *LICS*, pages 257–268, 2015.

[CY88] Costas Courcoubetis and Mihalis Yannakakis. Verifying temporal properties of finite-state probabilistic programs. In *FOCS*, pages 338–345, 1988.

[CY95] C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *J. ACM*, 42(4):857–907, 1995.

[CZ11] Edmund M. Clarke and Paolo Zuliani. Statistical model checking for cyber-physical systems. In *ATVA*, pages 1–12, 2011.

[DA97] Luca De Alfaro. *Formal verification of probabilistic systems*. PhD thesis, 1997.

[dAFS04] Luca de Alfaro, Marco Faella, and Mariëlle Stoelinga. Linear and branching metrics for quantitative transition systems. In *ICALP*, pages 97–109, 2004.

[dAKN$^+$00] L. de Alfaro, M. Kwiatkowska, G. Norman, D. Parker, and R. Segala. Symbolic model checking of probabilistic processes using MTBDDs and the Kronecker representation. In *TACAS*, 2000.

[dAMRS07] Luca de Alfaro, Rupak Majumdar, Vishwanath Raman, and Mariëlle Stoelinga. Game relations and metrics. In *LICS*, pages 99–108, 2007.

[DDL$^+$12] Alexandre David, Dehui Du, Kim G. Larsen, Axel Legay, Marius Mikucionis, Danny Bøgsted Poulsen, and Sean Sedwards. Statistical model checking for stochastic hybrid systems. In *HSB*, pages 122–136, 2012.

[DDL$^+$13] Alexandre David, Dehui Du, Kim Guldstrand Larsen, Axel Legay, and Marius Mikucionis. Optimizing control strategy using statistical model checking. In *NASA Formal Methods*, pages 352–367, 2013.

[DGJP99] Josee Desharnais, Vineet Gupta, Radha Jagadeesan, and Prakash Panangaden. Metrics for labeled Markov systems. In *CONCUR*, pages 258–273, 1999.

[DGV99] Marco Daniele, Fausto Giunchiglia, and Moshe Y. Vardi. Improved automata generation for linear temporal logic. In *CAV*, pages 249–260, 1999.

[DH11] Yuxin Deng and Matthew Hennessy. Compositional reasoning for markov decision processes - (extended abstract). In *FSEN*, pages 143–157, 2011.

[DH13] Y. Deng and M. Hennessy. On the semantics of Markov automata. *Inf. Comput.*, 222:139–168, 2013.

[DHR08] Laurent Doyen, Thomas A. Henzinger, and Jean-François Raskin. Equivalence of labeled Markov chains. *Int. J. Found. Comput. Sci.*, 19(3):549–563, 2008.

[DJKV17] Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen, and Matthias Volk. A storm is coming: A modern probabilistic model checker. In *CAV*, pages 592–600, 2017.

[DJW$^+$14] C. Dehnert, N. Jansen, R. Wimmer, E. Ábrahám, and J.-P. Katoen. Fast debugging of PRISM models. In *ATVA*, pages 146–162, 2014.

[DK05] P.R. D'Argenio and J.-P. Katoen. A theory of stochastic systems, part I: Stochastic automata. *Inf. Comput.*, 203(1):1–38, 2005.

[DLL$^+$11a] Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikucionis, Danny Bøgsted Poulsen, Jonas van Vliet, and Zheng Wang. Statistical model checking for networks of priced timed automata. In *FORMATS*, 2011.

[DLL$^+$11b] Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikucionis, and Zheng Wang. Time for statistical model checking of real-time systems. In *CAV*, pages 349–355, 2011.

[DLL+15] Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikucionis, and Danny Bøgsted Poulsen. Uppaal SMC tutorial. *STTT*, 17(4):397–415, 2015.

[DLLF+16] Alexandre Duret-Lutz, Alexandre Lewkowicz, Amaury Fauchille, Thibaud Michaud, Etienne Renault, and Laurent Xu. Spot 2.0 — a framework for LTL and ω-automata manipulation. In *ATVA*, pages 122–129, October 2016.

[DMS14a] Laurent Doyen, Thierry Massart, and Mahsa Shirmohammadi. Limit synchronization in markov decision processes. In *FoSSaCS*, volume 8412 of *LNCS*, pages 58–72. Springer, 2014.

[DMS14b] Laurent Doyen, Thierry Massart, and Mahsa Shirmohammadi. Robust synchronization in markov decision processes. In *CONCUR*, volume 8704 of *LNCS*, pages 234–248. Springer, 2014.

[EGF12] Christian Ellen, Sebastian Gerwinn, and Martin Fränzle. Confidence bounds for statistical model checking of probabilistic hybrid systems. In *FORMATS*, pages 123–138, 2012.

[EH82] E. A. Emerson and J. Y. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. pages 169–180, 1982.

[EH00] Kousha Etessami and Gerard J. Holzmann. Optimizing Büchi automata. In *CONCUR*, pages 153–167, 2000.

[EHK+13] C. Eisentraut, H. Hermanns, J. Krämer, A. Turrini, and L. Zhang. Deciding bisimilarities on distributions. In *QEST*, pages 72–88, 2013.

[EHZ10] C. Eisentraut, H. Hermanns, and L. Zhang. On probabilistic automata in continuous time. In *LICS*, pages 342–351, 2010.

[EK14] Javier Esparza and Jan Křetínský. From LTL to deterministic automata: A Safraless compositional approach. In *CAV*, pages 192–208, 2014.

[EKM06] J. Esparza, A. Kučera, and R. Mayr. Model-checking probabilistic pushdown automata. 2(1:2):1–31, 2006.

[EKVY08] K. Etessami, M. Kwiatkowska, M. Vardi, and M. Yannakakis. Multi-objective model checking of Markov decision processes. *LMCS*, 4(4):1–21, 2008.

[EY09] Kousha Etessami and Mihalis Yannakakis. Recursive markov chains, stochastic grammars, and monotone systems of nonlinear equations. *J. ACM*, 56(1):1:1–1:66, 2009.

[FK15] V. Forejt and J. Krčál. On frequency LTL in probabilistic systems. In *CONCUR*, volume 42 of *LIPIcs*. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

[FKR95] J.A. Filar, D. Krass, and K.W Ross. Percentile performance criteria for limiting average Markov decision processes. *Automatic Control, IEEE Transactions on*, 40(1):2–10, Jan 1995.

[FKS16] Nathanaël Fijalkow, Stefan Kiefer, and Mahsa Shirmohammadi. Trace refinement in labelled Markov decision processes. In *FOSSACS*, pages 303–318, 2016.

[FL79] M. Fischer and R. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 18:194–211, 1979.

[FL09] Oliver Friedmann and Martin Lange. Solving parity games in practice. In *ATVA*, pages 182–196, 2009.

[FL15] Dana Fisman and Yoad Lustig. A modular approach for büchi determinization. In *CONCUR*, pages 368–382, 2015.

[FPP04] Norm Ferns, Prakash Panangaden, and Doina Precup. Metrics for finite Markov decision processes. In *IAAI*, pages 950–951, 2004.

[Fri03] Carsten Fritz. Constructing Büchi automata from linear temporal logic using simulation relations for alternating Büchi automata. In *CIAA*, pages 35–48, 2003.

[FZ14] Yuan Feng and Lijun Zhang. When equivalence and bisimulation join forces in probabilistic automata. In *FM*, volume 8442 of *LNCS*, pages 247–262. Springer, 2014.

[GA12] S. Georgievska and S. Andova. Probabilistic may/must testing: retaining probabilities by restricted schedulers. *Formal Asp. Comput.*, 24(4-6):727–748, 2012.

[GDF09] S. Giro, P.R. D'Argenio, and L.M. Ferrer Fioriti. Partial order reduction for probabilistic systems: A revision for distributed schedulers. In *CONCUR*, pages 338–353, 2009.

[Gil57] D. Gillette. Stochastic games with zero stop probabilities. In M. Dresher, A.W. Tucker, and P. Wolfe, editors, *Contributions to the Theory of Games III*, number 39 in Annals of Mathematical Studies, chapter 9, page 179–187. Princeton University Press, Princeton, 1957.

[GL02] Dimitra Giannakopoulou and Flavio Lerda. From states to transitions: Improving translation of LTL formulae to Büchi automata. In *FORTE*, pages 308–326, 2002.

[GO01] Paul Gastin and Denis Oddoux. Fast LTL to Büchi automata translation. In *CAV*, pages 53–65, 2001. Tool accessible at `http://www.lsv.ens-cachan.fr/~gastin/ltl2ba/`.

[GP11] Antoine Girard and George J. Pappas. Approximate bisimulation: A bridge between computer science and control theory. *Eur. J. Control*, 17(5-6):568–578, 2011.

[GWX17] Hugo Gilbert, Paul Weng, and Yan Xu. Optimizing quantiles in preference-based markov decision processes. In *AAAI*, pages 3569–3575, 2017.

[Hen12] Matthew Hennessy. Exploring probabilistic bisimulations, part I. *Formal Asp. Comput.*, 24(4-6):749–768, 2012.

[HG16] Yonghui Huang and Xianping Guo. Minimum average value-at-risk for finite horizon semi-markov decision processes in continuous time. *SIAM Journal on Optimization*, 26(1):1–28, 2016.

[HHWZ10] Ernst Moritz Hahn, Holger Hermanns, Björn Wachter, and Lijun Zhang. PASS: abstraction refinement for infinite probabilistic models. In *TACAS*, pages 353–357, 2010.

[HJ94] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. 6:512–535, 1994.

[HJB$^+$10] Ru He, Paul Jennings, Samik Basu, Arka P. Ghosh, and Huaiqing Wu. A bounded statistical approach for model checking of unbounded until properties. In *ASE*, pages 225–234, 2010.

[HK97] Michael Huth and Marta Z. Kwiatkowska. Quantitative analysis and model checking. In *LICS*, pages 111–122. IEEE Computer Society, 1997.

[HK15] Christoph Haase and Stefan Kiefer. The odds of staying on budget. In *ICALP*, pages 234–246, 2015.

[HKK13] Holger Hermanns, Jan Krčál, and Jan Křetínský. Compositional verification and optimization of interactive Markov chains. In *CONCUR*, pages 364–379, 2013.

[HKL17] Christoph Haase, Stefan Kiefer, and Markus Lohrey. Computing quantiles in markov chains with multi-dimensional costs. In *LICS*, pages 1–12, 2017.

[HKM06] T. Henzinger, O. Kupferman, and R. Majumdar. On the universal and existential fragments of the μ-calculus. *Theoretical Computer Science*, 354(2):173–186, 2006.

[HKN$^+$03] H. Hermanns, M. Kwiatkowska, G. Norman, D. Parker, and M. Siegle. On the use of MTBDDs for performability analysis and verification of stochastic systems. *Journal of Logic and Algebraic Programming: Special Issue on Probabilistic Techniques for the Design and Analysis of Systems*, 56(1-2):23–67, 2003.

[HLMP04] Thomas Hérault, Richard Lassaigne, Frédéric Magniette, and Sylvain Peyronnet. Approximate probabilistic model checking. In *VMCAI*, pages 73–84, 2004.

[HLS$^+$15] Ernst Moritz Hahn, Guangyuan Li, Sven Schewe, Andrea Turrini, and Lijun Zhang. Lazy probabilistic model checking without determinisation. In *CONCUR*, volume 42 of *LIPIcs*, pages 354–367, 2015.

[HM14] Serge Haddad and Benjamin Monmege. Reachability in MDPs: Refining convergence of value iteration. In *International Workshop on Reachability Problems*, pages 125–137. Springer, 2014.

[HMW09] Thomas A. Henzinger, Maria Mateescu, and Verena Wolf. Sliding window abstraction for infinite markov chains. In *CAV*, volume 5643 of *LNCS*, pages 337–352. Springer, 2009.

[HMZ$^+$12] David Henriques, João Martins, Paolo Zuliani, André Platzer, and Edmund M. Clarke. Statistical model checking for Markov decision processes. In *QEST*, pages 84–93, 2012.

[How60] H. Howard. *Dynamic Programming and Markov Processes*. MIT Press, 1960.

[HS86] S. Hart and M. Sharir. Probabilistic propositional temporal logics. *Information and Control*, 70(2/3):97–155, 1986.

[HSAHB99] J. Hoey, R. St-Aubin, A. Hu, and C. Boutilier. Spudd: Stochastic planning using decision diagrams. In *In Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pages 279–288. Morgan Kaufmann, 1999.

[HWZ08] Holger Hermanns, Björn Wachter, and Lijun Zhang. Probabilistic CEGAR. In *Computer Aided Verification, 20th International Conference, CAV 2008, Princeton, NJ, USA, July 7-14, 2008, Proceedings*, pages 162–175, 2008.

[JBB$^+$17] Swen Jacobs, Nicolas Basset, Roderick Bloem, Romain Brenguier, Maximilien Colange, Peter Faymonville, Bernd Finkbeiner, Ayrat Khalimov, Felix Klein, Thibaud Michaud, Guillermo A. Pérez, Jean-François Raskin, Ocan Sankur, and Leander Tentrup. The 4th reactive synthesis competition (SYNTCOMP 2017): Benchmarks, participants & results. *CoRR*, abs/1711.11439, 2017.

[JCL$^+$09] Sumit Kumar Jha, Edmund M. Clarke, Christopher James Langmead, Axel Legay, André Platzer, and Paolo Zuliani. A bayesian approach to model checking biological systems. In *CMSB*, pages 218–234, 2009.

[JLS12] Cyrille Jégourel, Axel Legay, and Sean Sedwards. A platform for high performance statistical model checking - PLASMA. In *TACAS*, pages 498–503, 2012.

[JMLM14] Manfred Jaeger, Hua Mao, Kim G. Larsen, and Radu Mardare. Continuity properties of distances for Markov processes. In *QEST*, pages 297–312, 2014.

[KE12] Jan Křetínský and Javier Esparza. Deterministic automata for the (F,G)-fragment of LTL. In P. Madhusudan and Sanjit A. Seshia, editors, *CAV*, volume 7358 of *Lecture Notes in Computer Science*, pages 7–22. Springer, 2012.

[KFKT11] Masayuki Kageyama, Takayuki Fujii, Koji Kanefuji, and Hiroe Tsubaki. Conditional value-at-risk for random immediate reward variables in markov decision processes. *American J. Computational Mathematics*, 1(3):183–188, 2011.

[KHW94] N. Kushmerick, S. Hanks, and D. Weld. An algorithm for probabilistic least-commitment planning. In *Proceedings of AAAI-94*, pages 1073–1078, 1994.

[KK99] M. Kearns and D. Koller. Efficient reinforcement learning in factored MDPs. IJCAI, pages 740–747, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[KKNP10] Mark Kattenbelt, Marta Z. Kwiatkowska, Gethin Norman, and David Parker. A game-based abstraction-refinement framework for markov decision processes. *Formal Methods in System Design*, 36(3):246–280, 2010.

[KL83] S. Kraus and D. J. Lehmann. Decision procedures for time and chance (extended abstract). In *FOCS*, pages 202–209, 1983.

[Kle] Joachim Klein. ltl2dstar - LTL to deterministic Streett and Rabin automata. `http://www.ltl2dstar.de/`.

[KLG13] Jan Křetínský and Ruslán Ledesma-Garza. Rabinizer 2: Small deterministic automata for LTL$\backslash$GU. In *ATVA*, pages 446–450, 2013.

[KNP02] M. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic symbolic model checker. In *TOOLS' 02*, pages 200–204, 2002.

[KNP12] M. Kwiatkowska, G. Norman, and D. Parker. The PRISM benchmark suite. In *QEST*, pages 203–204, 2012.

[KP99] D. Koller and R. Parr. Computing factored value functions for policies in structured MDPs. In *In Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 1332–1339. Morgan Kaufmann, 1999.

[KP13] Marta Kwiatkowska and David Parker. Automated verification and strategy synthesis for probabilistic systems. In *Automated Technology for Verification and Analysis*, pages 5–22. Springer, 2013.

[KPV06] Orna Kupferman, Nir Piterman, and Moshe Y. Vardi. Safraless compositional synthesis. In *CAV*, volume 4144 of *LNCS*, pages 31–44. Springer, 2006.

[KS08] Antonín Kucera and Oldrich Strazovský. On the controller synthesis for finite-state markov decision processes. *Fundam. Inform.*, 82(1-2):141–153, 2008.

[KV00] O. Kupferman and M.Y. Vardi. An automata-theoretic approach to modular model checking. 22:87–128, 2000.

[KV15] Dileep Kini and Mahesh Viswanathan. Limit deterministic and probabilistic automata for LTL $\setminus$ GU. In *TACAS*, pages 628–642, 2015.

[KV17] Dileep Kini and Mahesh Viswanathan. Optimal translation of LTL to limit deterministic automata. In *TACAS 2017*, 2017. To appear.

[KVAK10] V.A. Korthikanti, M. Viswanathan, G. Agha, and Y. Kwon. Reasoning about MDPs as transformers of probability distributions. In *QEST*, pages 199–208, 2010.

[KZH$^+$11] Joost-Pieter Katoen, Ivan S. Zapreev, Ernst Moritz Hahn, Holger Hermanns, and David N. Jansen. The ins and outs of the probabilistic model checker MRMC. *Perform. Eval.*, 68(2):90–104, 2011.

[Lar12] Kim G. Larsen. Statistical model checking, refinement checking, optimization, ... for stochastic hybrid systems. In *FORMATS*, pages 7–10, 2012.

[Lar13] Kim Guldstrand Larsen. Priced timed automata and statistical model checking. In *IFM*, 2013.

[LL08] Jianwei Li and Weiyi Liu. A novel heuristic q-learning algorithm for solving stochastic games. In *IJCNN*, pages 1135–1144, 2008.

[LL13] F. Leitner-Fischer and S. Leue. Probabilistic fault tree synthesis using causality computation. *IJCCBS*, 4(2):119–143, 2013.

[LP08] Richard Lassaigne and Sylvain Peyronnet. Probabilistic verification and approximation. *Ann. Pure Appl. Logic*, 152(1-3):122–131, 2008.

[LP12] Richard Lassaigne and Sylvain Peyronnet. Approximate planning and verification for large Markov decision processes. In *SAC*, pages 1314–1319, 2012.

[LS83] D. J. Lehmann and S. Shelah. Reasoning with time and chance (extended abstract). In *ICALP*, pages 445–457, 1983.

[LS89] Kim G. Larsen and Arne Skou. Bisimulation through probabilistic testing. In *POPL*, pages 344–352, 1989.

[LST14] Axel Legay, Sean Sedwards, and Louis-Marie Traonouez. Scalable verification of markov decision processes. In *SEFM*, pages 350–362, 2014.

[LZB17] Xiaocheng Li, Huaiyang Zhong, and Margaret L. Brandeau. Quantile markov decision process. *CoRR*, abs/1711.05788, 2017.

[M$^+$74] R.M. May et al. Biological populations with nonoverlapping generations: stable points, stable cycles, and chaos. *Science*, 186(4164):645–647, 1974.

[Mar06] A.A. Markov. Rasprostranenie zakona bol'shih chisel na velichiny, zavisyaschie drug ot druga. *Izvestiya Fiziko-matematicheskogo obschestva pri Kazanskom universitete*, 15(94):135–156, 1906.

[ML16] Philipp J. Meyer and Michael Luttenberger. Solving mean-payoff games on the GPU. In *ATVA*, pages 262–267, 2016.

[MLG05] H Brendan McMahan, Maxim Likhachev, and Geoffrey J Gordon. Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. In *ICML*, 2005.

[MP04] A. Miner and D. Parker. *Validation of Stochastic Systems: A Guide to Current Research*, volume 2925 of *Lecture Notes in*

Computer Science (Tutorial Volume), chapter Symbolic Representations and Analysis of Large Probabilistic Systems, pages 296–338. Springer, 2004.

[MSL18] Philipp J. Meyer, Salomon Sickert, and Michael Luttenberger. Strix: Explicit reactive synthesis strikes back! In *CAV (1)*, volume 10981 of *Lecture Notes in Computer Science*, pages 578–586. Springer, 2018.

[MY17] Christopher W. Miller and Insoon Yang. Optimal control of conditional value-at-risk in continuous time. *SIAM J. Control and Optimization*, 55(2):856–884, 2017.

[Nor98] James R Norris. *Markov chains*. Cambridge university press, 1998.

[NPS13] Gethin Norman, David Parker, and Jeremy Sproston. Model checking for probabilistic timed automata. *Formal Methods in System Design*, 43(2):164–190, 2013.

[Par81] D.M.R. Park. Concurrency and automata on infinite sequences. In *Theor. Comp. Sci.*, pages 167–183, 1981.

[PGL$^+$13] Sucheendra K. Palaniappan, Benjamin M. Gyori, Bing Liu, David Hsu, and P. S. Thiagarajan. Statistical model checking based calibration and analysis of bio-pathway models. In *CMSB*, pages 120–134, 2013.

[Pit06] Nir Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. In *LICS*, pages 255–264, 2006.

[Pnu77] Amir Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57, 1977.

[PP06] Nir Piterman and Amir Pnueli. Faster solutions of Rabin and Streett games. In *LICS*, pages 275–284, 2006.

[Put94] M.L. Puterman. *Markov Decision Processes*. John Wiley and Sons, 1994.

[Pye03] Larry D. Pyeatt. Reinforcement learning with decision trees. In *The 21st IASTED International Multi-Conference on Applied Informatics (AI 2003), February 10-13, 2003, Innsbruck, Austria*, pages 26–31, 2003.

[RP09] Diana El Rabih and Nihal Pekergin. Statistical model checking using perfect simulation. In *ATVA*, pages 120–134, 2009.

[RRS17] Mickael Randour, Jean-François Raskin, and Ocan Sankur. Percentile queries in multi-dimensional markov decision processes. *Formal Methods in System Design*, 50(2-3):207–248, 2017.

[RU00] R. Tyrrell Rockafellar and Stanislav Uryasev. Optimization of conditional value-at-risk. *Journal of Risk*, 2:21–41, 2000.

[Saf88] Shmuel Safra. On the complexity of omega-automata. In *FOCS*, pages 319–327, 1988.

[SB00] Fabio Somenzi and Roderick Bloem. Efficient Büchi automata from LTL formulae. In *CAV*, pages 248–263, 2000.

[Sch09] Sven Schewe. Tighter bounds for the determinisation of Büchi automata. In *FoSSaCS*, volume 5504 of *LNCS*, pages 167–181, 2009.

[Sha53] L. S. Shapley. Stochastic games. *Proc. Nat. Acad. Sci. U.S.A.*, 39:1095–1100, 1953.

[SKC$^+$15] Maria Svoreňová, Jan Křetínský, Martin Chmelík, Krishnendu Chatterjee, Ivana Černá, and Calin Belta. Temporal logic control for stochastic linear systems using abstraction refinement of probabilistic games. In *HSCC*, pages 259–268, 2015.

[SL94] R. Segala and N.A. Lynch. Probabilistic simulations for probabilistic processes. In *CONCUR*, pages 481–496, 1994.

[SLT10] C. S. Raghavendra S. Liu, A. Panangadan and A. Talukder. Compact representation of coordinated sampling policies for body sensor networks. In *In Proceedings of Workshop on Advances in Communication and Networks (Smart Homes for Tele-Health)*, pages 6–10. IEEE, 2010.

[SLW$^+$06] Alexander L. Strehl, Lihong Li, Eric Wiewiora, John Langford, and Michael L. Littman. PAC model-free reinforcement learning. In *ICML*, pages 881–888, 2006.

[SPK13] G. Shani, J. Pineau, and R. Kaplow. A survey of point-based pomdp solvers. *AAMAS*, 27(1):1–51, 2013.

[SR11] D. Sangiorgi and J. Rutten. *Advanced Topics in Bisimulation and Coinduction*. Cambridge University Press, New York, NY, USA, 1st edition, 2011.

[SVA04] Koushik Sen, Mahesh Viswanathan, and Gul Agha. Statistical model checking of black-box probabilistic systems. In *CAV*, pages 202–215, 2004.

[SVA05] Koushik Sen, Mahesh Viswanathan, and Gul Agha. On statistical model checking of stochastic systems. In *CAV*, pages 266–280, 2005.

[SZG12] L. Song, L. Zhang, and J. C. Godskesen. Late weak bisimulation for Markov automata. *CoRR*, abs/1202.4116, 2012.

[TD14] Cong Tian and Zhenhua Duan. Buchi determinization made tighter. Technical Report abs/1404.1436, arXiv.org, 2014.

[THHY12] Takashi Tomita, Shin Hiura, Shigeki Hagihara, and Naoki Yonezaki. A temporal logic with mean-payoff constraints. In *ICFEM*, volume 7635 of *LNCS*. Springer, 2012.

[THY11] Takashi Tomita, Shigeki Hagihara, and Naoki Yonezaki. A probabilistic temporal logic with frequency operators and its model checking. In *INFINITY*, 2011.

[TT16] Alain Tcheukam and Hamidou Tembine. One swarm per queen: A particle swarm learning for stochastic games. In *SASO*, pages 144–145, 2016.

[UB13] Michael Ummels and Christel Baier. Computing quantiles in markov reward models. In *FOSSACS*, pages 353–368, 2013.

[Ujm15] Mateusz Ujma. *On Verification and Controller Synthesis for Probabilistic Systems at Runtime*. PhD thesis, Wolfson College, Oxford, 2015.

[Val84] Leslie G. Valiant. A theory of the learnable. In *STOC*, pages 436–445. ACM, 1984.

[vBSW07] F. van Breugel, B. Sharma, and J. Worrell. Approximating a behavioural pseudometric without discount for probabilistic systems. In *FOSSACS*, pages 123–137, 2007.

[vBW06] Franck van Breugel and James Worrell. Approximating and computing behavioural distances in probabilistic transition systems. *Theor. Comput. Sci.*, 360(1-3):373–385, 2006.

[VW86] Moshe Y. Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *LICS*, pages 332–344, 1986.

[Wat89] C.J.C.H. Watkins. *Learning from Delayed Rewards*. PhD thesis, Cambridge University, 1989.

[WBB$^+$10] Ralf Wimmer, Bettina Braitling, Bernd Becker, Ernst Moritz Hahn, Pepijn Crouzen, Holger Hermanns, Abhishek Dhama, and Oliver E. Theel. Symblicit calculation of long-run averages for concurrent probabilistic systems. In *QEST*, pages 27–36, 2010.

[WJV$^+$13] Ralf Wimmer, Nils Jansen, Andreas Vorpahl, Erika Ábrahám, Joost-Pieter Katoen, and Bernd Becker. High-level counterexamples for probabilistic automata. In *QEST*, pages 39–54, 2013.

[WT16] Min Wen and Ufuk Topcu. Probably approximately correct learning in stochastic games with temporal logic specifications. In *IJCAI*, pages 3630–3636, 2016.

[YCZ10] Håkan L. S. Younes, Edmund M. Clarke, and Paolo Zuliani. Statistical verification of probabilistic properties with unbounded until. In *SBMF*, pages 144–160, 2010.

[YKNP06] Håkan L. S. Younes, Marta Z. Kwiatkowska, Gethin Norman, and David Parker. Numerical vs. statistical probabilistic model checking. *STTT*, 8(3):216–228, 2006.

[YS02] Håkan L. S. Younes and Reid G. Simmons. Probabilistic verification of discrete event systems using acceptance sampling. In *CAV*, pages 223–235. Springer, 2002.

[ZPC10] Paolo Zuliani, André Platzer, and Edmund M. Clarke. Bayesian statistical model checking with application to simulink/stateflow verification. In *HSCC*, pages 243–252, 2010.

[ZSF12] Zahra Zamani, Scott Sanner, and Cheng Fang. Symbolic dynamic programming for continuous state and action mdps. In *AAAI*, 2012.

PART II

SELECTED PAPERS

A Note on Copyright: Acknowledgment of the Publishers

Due to copyright reasons, we list the complete bibliographic information on all attached preprints of the papers, acknowledging the original source of the publications below:

- Holger Hermanns, Jan Krčál, and Jan Křetínský. Probabilistic bisimulation: Naturally on distributions. 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 249–265. Springer, 2014.

  `http://dx.doi.org/10.1007/978-3-662-44584-6_18`

- Tomáš Brázdil, Krishnendu Chatterjee, Martin Chmelík, Vojtěch Forejt, Jan Křetínský, Marta Z. Kwiatkowska, David Parker, and Mateusz Ujma. Verification of Markov decision processes using learning algorithms. In Franck Cassez and Jean-François Raskin, editors, *Automated Technology for Verification and Analysis - 12th International Symposium, ATVA 2014, Sydney, NSW, Australia, November 3-7, 2014, Proceedings*, volume 8837 of *Lecture Notes in Computer Science*, pages 98–114. Springer, 2014.

  `http://dx.doi.org/10.1007/978-3-319-11936-6_8`

- Tomáš Brázdil, Krishnendu Chatterjee, Martin Chmelík, Andreas Fellner, and Jan Křetínský. Counterexample explanation by learning small strategies in Markov decision processes. In Daniel Kroening and Corina S. Pasareanu, editors, *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I*, volume 9206 of *Lecture Notes in Computer Science*, pages 158–177. Springer, 2015.

  `http://dx.doi.org/10.1007/978-3-319-11936-6_8`

- Krishnendu Chatterjee, Zuzana Křetínská, and Jan Křetínský. Unifying two views on multiple mean-payoff objectives in Markov decision processes. *Logical Methods in Computer Science*, 13(2), 2017.

  `https://doi.org/10.23638/LMCS-13(2:15)2017`

- Przemyslaw Daca, Thomas A. Henzinger, Jan Křetínský, and Tatjana Petrov. Faster statistical model checking for unbounded temporal properties. *ACM Trans. Comput. Log.*, 18(2):12:1–12:25, 2017.

`http://doi.acm.org/10.1145/3060139`

- Javier Esparza, Jan Křetínský, Jean-François Raskin, and Salomon Sickert. From LTL and limit-deterministic Büchi automata to deterministic parity automata. In Axel Legay and Tiziana Margaria, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part I*, volume 10205 of *Lecture Notes in Computer Science*, pages 426–442. Springer, 2017.

  `https://doi.org/10.1007/978-3-662-54577-5_25`

- Javier Esparza, Jan Křetínský, and Salomon Sickert. One theorem to rule them all: A unified translation of LTL into ω-automata. 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 384–393. ACM, 2018.

  `http://doi.acm.org/10.1145/3209108.3209161`

- Jan Křetínský and Tobias Meggendorfer. Conditional value-at-risk for reachability and mean payoff in Markov decision processes. 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 609–618. ACM, 2018.

  `http://doi.acm.org/10.1145/3209108.3209176`

- Edon Kelmendi, Julia Krämer, Jan Křetínský, and Maximilian Weininger. Value iteration for simple stochastic games: Stopping criterion and learning algorithm. In Hana Chockler and Georg Weissenbacher, editors, *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I*, volume 10981 of *Lecture Notes in Computer Science*, pages 623–642. Springer, 2018.

  `https://doi.org/10.1007/978-3-319-96145-3\_36`

- Jan Křetínský, Tobias Meggendorfer, Salomon Sickert, and Christopher Ziegler. Rabinizer 4: From LTL to your favourite deterministic automaton. In Hana Chockler and Georg Weissenbacher, editors,

Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I, volume 10981 of *Lecture Notes in Computer Science*, pages 567–577. Springer, 2018.

`https://doi.org/10.1007/978-3-319-96145-3\_30`

A Note on Contribution: Explanation of Assumptions

According to the MU Directive on Habilitation Procedures and Professor Appointment Procedures No.7/2017, we list the quantitative and qualitative (content-wise) contribution of the author. We interpret the qualitative contribution as the collection of the aspects listed in Table 5.1, considering these to be commonly accepted parts of the process of creating a paper in computer science. Respecting the tradition, we do not list, for instance, the aspect of acquiring funding to perform the given research, despite its scientific character and paramount effect on the existence of the paper.

In contrast, since a contribution can only be evaluated quantitatively under a given metric and the above-mentioned directive suggests no such metric and within tradition there is no agreement on it, we consider this task not well defined. Due to the lack of generally more acceptable criteria, we take the liberty of defining the contribution of an author within the scope of a paper with the alphabetical ordering as the reciprocal of the number of co-authors, due to the seeming canonicity of the formula and its exceptional blandness and insipidity.

Table 5.1: Author's contribution to the papers of the attached collection. Dark gray denotes major contribution, light gray denotes minor or no contribution, crosses denote non-applicability.

	conceptual		mathematical		programming		writing			
	topic	approach	idea	proofs	impleme-ntation	experi-ments	story	body	appendix	no. of authors
Probabilistic bisimulation: Naturally on distributions CONCUR 2014	□	■	■	■	✗	✗	■	■	■	3
Verification of Markov decision processes using learning algorithms ATVA 2014	■	□	■	■	□	□	■	■	■	8
Counterexample explanation by learning small strategies in Markov decision processes CAV 2015	□	□	■	■	□	■	■	■	✗	5
Unifying two views on multiple mean-payoff objectives in Markov decision processes, Log. Meth. in Comp. Sci. 2017	□	□	■	■	✗	✗	□	■	■	3
Faster statistical model checking for unbounded temporal properties, ACM Trans. on Comp. Log. 2017	■	■	■	■	□	□	■	■	■	4
From LTL and limit-deterministic Büchi automata to deterministic parity automata TACAS 2017	□	■	■	■	□	□	■	■	□	4
One theorem to rule them all: A unified translation of LTL into ω-automata LICS 2018	■	■	■	□	✗	✗	■	■	□	3
Conditional value-at-risk for reachability and mean payoff in Markov decision processes LICS 2018	□	■	■	■	✗	✗	■	■	□	2
Value iteration for simple stochastic games: Stopping criterion and learning algorithm CAV 2018	■	■	□	□	□	□	■	■	□	4
Rabinizer 4: From LTL to your favourite deterministic automaton CAV 2018	■	■	✗	✗	□	□	■	■	✗	4

Probabilistic Bisimulation: Naturally on Distributions[⋆]

Holger Hermanns[1], Jan Krčál[1], and Jan Křetínský[2]

[1] Saarland University – Computer Science, Saarbrücken, Germany
{hermanns,krcal}@cs.uni-saarland.de
[2] IST Austria jan.kretinsky@ist.ac.at

Abstract. In contrast to the usual understanding of probabilistic systems as stochastic processes, recently these systems have also been regarded as transformers of probabilities. In this paper, we give a *natural* definition of strong bisimulation for probabilistic systems corresponding to this view that treats probability *distributions* as first-class citizens. Our definition applies in the same way to discrete systems as well as to systems with uncountable state and action spaces. Several examples demonstrate that our definition refines the understanding of behavioural equivalences of probabilistic systems. In particular, it solves a longstanding open problem concerning the representation of memoryless continuous time by memoryfull continuous time. Finally, we give algorithms for computing this bisimulation not only for finite but also for classes of uncountably infinite systems.

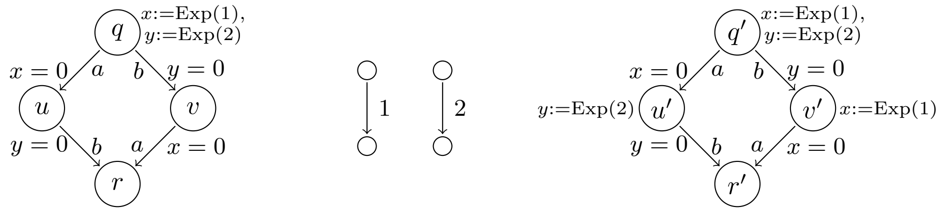
1 Introduction

Continuous time concurrency phenomena can be addressed in two principal manners: On the one hand, *timed automata* (TA) extend interleaving concurrency with real-valued clocks [2]. On the other hand, time can be represented by memoryless stochastic time, as in *continuous time Markov chains* (CTMC) and extensions, where time is represented in the form of exponentially distributed random delays [33,30,6,23]. TA and CTMC variations have both been applied to very many intriguing cases, and are supported by powerful real-time, respectively stochastic time model checkers [3,37] with growing user bases. The models are incomparable in expressiveness, but if one extends timed automata with the possibility to sample from exponential distributions [5,10,28], there appears to be a natural bridge from CTMC to TA. This kind of stochastic semantics of timed

⋆ This work is supported by the EU 7th Framework Programme under grant agreements 295261 (MEALS) and 318490 (SENSATION), European Research Council (ERC) under grant agreement 267989 (QUAREM), Austrian Science Fund (FWF) project S11402-N23 (RiSE), Czech Science Foundation under grant agreement P202/12/G061, the DFG Transregional Collaborative Research Centre SFB/TR 14 AVACS, and by the CAS/SAFEA International Partnership Program for Creative Research Teams. Jan Křetínský is currently on leave from Faculty of Informatics, Masaryk University, Czech Republic.

automata has recently gained considerable popularity by the statistical model checking approach to TA analysis [14,13].

Still there is a disturbing difference, and this difference is the original motivation [12] of the work presented in this paper. The obvious translation of an exponentially distributed delay into a clock expiration sampled from the very same exponential probability distribution fails in the presence of concurrency. This is because the translation is not fully compatible with the natural interleaving concurrency semantics for TA respectively CTMC. This is illustrated by the following example, which in the middle displays two small CTMC, which are supposed to run independently and concurrently.



On the left and right we see two stochastic automata (a variation of timed automata formally defined in Section 3). They have clocks x and y which are initialized by sampling from exponential distributions, and then each run down to 0. The first one reaching 0 triggers a transition and the other clock keeps on running unless resampled, which happens on the right, but not on the left. The left model is obtained by first translating the respective CTMC, and then applying the natural TA interleaving semantics, while the right model is obtained by first applying the equally natural CTMC interleaving semantics prior to translation.

The two models have subtly different semantics in terms of their underlying dense probabilistic timed transition systems. This can superficially be linked to the memoryless property of exponential distributions, yet there is no formal basis for proving equivalence. Our paper closes this gap, which has been open for at least 15 years, by introducing a natural *continuous-space distribution-based* bisimulation. Our result is embedded in several further intriguing application contexts and algorithmic achievements for this novel bisimulation.

The theory of bisimulations is a well-established and elegant framework to describe equivalence between processes based on their behaviour. In the standard semantics of probabilistic systems [38,45], when a probabilistic step from a state to a distribution is taken, the random choice is resolved and we instead continue from one of the successor states. Recently, there has been considerable interest in instead regarding probabilistic systems as deterministic transformers of probability *distributions* [36,1,20], where the choice is not resolved and we continue from the distribution over successors. Thus, instead of the current state the transition changes the current distribution over the states. Although the distribution semantics is very natural in many contexts [29], it has been only partially reflected in the study of bisimulations [29,19,24,23].

Our definition arises as an unusual, but very simple instantiation of the standard coalgebraic framework for bisimulations [42]. (No knowledge of coalgebra is required from the reader though.) Despite its simplicity, the resulting notion is surprisingly fruitful, not only because it indeed solves the longstanding correspondence problem between CTMC and TA with stochastic semantics.

Firstly, it is more adequate than other equivalences when applied to systems with distribution semantics, including large-population models where different parts of the population act differently [39]. Indeed, as argued in [26], some equivalent states are not identified in the standard probabilistic bisimulations and too many are identified in the recent distribution based bisimulations [19,24]. Our approach allows for a bisimulation identifying precisely the desired states [26].

Secondly, our bisimulation over distributions induces an equivalence *on states*, and this relation equates behaviourally indistinguishable states which in many settings are unnecessarily distinguished by standard bisimulations. We shall discuss this phenomenon in the context of several applications. Nevertheless, the key idea to work with distributions instead of single states also bears disadvantages. The main difficulty is that even for finite systems the space of distributions is uncountable, thus bisimulation is difficult to compute. However, we show that it admits a concise representation using methods of linear algebra and we provide an algorithm for computing it. Further, in order to cover e.g. continuous-time systems, we need to handle both uncountably many states (that store the sampled time) and labels (real time durations). Fortunately, there is an elegant way to do so using the standard coalgebra framework. Moreover, it can easily be further generalized, e.g. adding rewards to the generic definition is a trivial task. **Our contribution** is the following:

- We give a natural definition of bisimulation from the distribution perspective for systems with generally uncountable spaces of states and labels.
- We argue by means of several applications that the definition can be considered more useful than the classical notions of probabilistic bisimulation.
- We provide an algorithm to compute this distributional bisimulation on finite non-deterministic probabilistic systems, and present a decision algorithm for uncountable continuous-time systems induced by the stochastic automata mentioned above.

A full version of this paper is available [31].

2 Probabilistic bisimulation on distributions

A (potentially uncountable) set S is a *measurable space* if it is equipped with a σ-algebra, denoted by $\Sigma(S)$. The elements of $\Sigma(S)$ are called *measurable sets.* For a measurable space S, let $\mathcal{D}(S)$ denote the set of *probability measures* (or *probability distributions*) over S. The following definition is similar to the treatment of [52].

Definition 1. *A* non-deterministic labelled Markov process *(NLMP) is a tuple* $\mathbf{P} = (S, L, \{\tau_a \mid a \in L\})$ *where S is a measurable space of* states*, L is a measur-*

able space of labels*, and $\tau_a : S \to \Sigma(\mathcal{D}(S))$ assigns to each state s a measurable set of probability measures $\tau_a(s)$ available in s under a.*[(1)]

When in a state $s \in S$, NLMP reads a label $a \in L$ and *non-deterministically* chooses a successor distribution $\mu \in \mathcal{D}(S)$ that is in the set of convex combinations[(2)] over $\tau_a(s)$, denoted by $s \xrightarrow{a} \mu$. If there is no such distribution, the process halts. Otherwise, it moves into a successor state according to μ. Considering convex combinations is necessary as it gives more power than pure resolution of non-determinism [43].

Example 1. If all sets are finite, we obtain *probabilistic automata (PA)* defined [43] as a triple $(S, L, \longrightarrow)$ where $\longrightarrow \subseteq S \times L \times \mathcal{D}(S)$ is a probabilistic transition relation with $(s, a, \mu) \in \longrightarrow$ if $\mu \in \tau_a(s)$.

Example 2. In the continuous setting, consider a random number generator that also remembers the previous number. We set $L = [0,1]$, $S = [0,1] \times [0,1]$ and $\tau_x(\langle \mathit{new}, \mathit{last} \rangle) = \{\mu_x\}$ for $x = \mathit{new}$ and $\emptyset$ otherwise, where μ_x is the uniform distribution on $[0,1] \times \{x\}$. If we start with a uniform distribution over S, the measure of successors under any $x \in L$ is 0. Thus in order to get any information of the system we have to consider successors under sets of labels, e.g. intervals.

For a measurable set $A \subseteq L$ of labels, we write $s \xrightarrow{A} \mu$ if $s \xrightarrow{a} \mu$ for some $a \in A$, and denote by $S_A := \{s \mid \exists \mu : s \xrightarrow{A} \mu\}$ the set of states having some outgoing label from A. Further, we can lift this to probability distributions by setting $\mu \xrightarrow{A} \nu$ if $\mu(S_A) > 0$ and $\nu = \frac{1}{\mu(S_A)} \int_{s \in S_A} \nu_s \, \mu(d\,s)$ for some measurable function assigning to each state $s \in S_A$ a measure ν_s such that $s \xrightarrow{A} \nu_s$. Intuitively, in μ we restrict to states that do not halt under A and consider all possible combinations of their transitions; we scale up by $\frac{1}{\mu(S_A)}$ to obtain a distribution again.

Example 3. In the previous example, let υ be the uniform distribution. Due to the independence of the random generator on previous values, we get $\upsilon \xrightarrow{[0,1]} \upsilon$. Similarly, $\upsilon \xrightarrow{[0.1,0.2]} \upsilon_{[0.1,0.2]}$ where $\upsilon_{[0.1,0.2]}$ is uniform on $[0,1]$ in the first component and uniform on $[0.1, 0.2]$ in the second component, with no correlation.

Using this notation, a non-deterministic and probabilistic system such as NLMP can be regarded as a non-probabilistic, thus solely non-deterministic, labelled transition system over the uncountable space of probability distributions. The natural bisimulation from this distribution perspective is as follows.

Definition 2. *Let $(S, L, \{\tau_a \mid a \in L\})$ be a NLMP and $R \subseteq \mathcal{D}(S) \times \mathcal{D}(S)$ be a symmetric relation. We say that R is a (strong)* probabilistic bisimulation *if for each $\mu \, R \, \nu$ and measurable $A \subseteq L$*

(1) We further require that for each $s \in S$ we have $\{(a, \mu) | \mu \in \tau_a(s)\} \in \Sigma(L) \otimes \Sigma(\mathcal{D}(S))$ and for each $A \in \Sigma(L)$ and $Y \in \Sigma(\mathcal{D}(S))$ we have $\{s \in S \mid \exists a \in A. \tau_a(s) \cap Y \neq \emptyset\} \in \Sigma(S)$. Here $\Sigma(\mathcal{D}(S))$ is the Giry σ-algebra [27] over $\mathcal{D}(X)$.

(2) A distribution $\mu \in \mathcal{D}(S)$ is a *convex combination* of a set $M \in \Sigma(\mathcal{D}(S))$ of distributions if there is a measure ν on $\mathcal{D}(S)$ such that $\nu(M) = 1$ and $\mu = \int_{\mu' \in \mathcal{D}(S)} \mu' \nu(d\mu')$.

1. $\mu(S_A) = \nu(S_A)$, and
2. for each $\mu \xrightarrow{A} \mu'$ there is a $\nu \xrightarrow{A} \nu'$ such that $\mu' \, R \, \nu'$.

We set $\mu \sim \nu$ if there is a probabilistic bisimulation R such that $\mu \, R \, \nu$.

Example 4. Considering Example 2, states $\{x\} \times [0, 1]$ form a class of $\sim$ for each $x \in [0, 1]$ as the old value does not affect the behaviour. More precisely, $\mu \sim \nu$ iff marginals of their first component are the same.

Naturalness. Our definition of bisimulation is not created ad-hoc as it often appears for relational definitions, but is actually an instantiation of the standard bisimulation for a particular *coalgebra*. Although this aspect is not necessary for understanding the paper, it is another argument for naturalness of our definition. For reader's convenience, we present a short introduction to coalgebras and the formal definitions in [31]. Here we only provide an intuitive explanation by example.

Non-deterministic labelled transition systems are essentially given by the transition function $S \to \mathcal{P}(S)^L$; given a state $s \in S$ and a label $a \in L$, we can obtain the set of the successors $\{s' \in S \mid s \xrightarrow{a} s'\}$. The transition function corresponds to a coalgebra, which induces a bisimulation coinciding with the classical one of Park and Milner [40]. Similarly, PA are given by the transition function $S \to \mathcal{P}(\mathcal{D}(S))^L$; instead of successors there are distributions over successors. Again, the corresponding coalgebraic bisimulation coincides with the classical ones of Larsen and Skou [38] and Segala and Lynch [44].

In contrast, our definition can be obtained by considering states S' to be distributions in $\mathcal{D}(S)$ over the original state space and defining the transition function to be $S' \to ([0, 1] \times \mathcal{P}(S'))^{\Sigma(L)}$. The difference to the standard non-probabilistic case is twofold: firstly, we consider all measurable sets of labels, i.e. all elements of $\Sigma(L)$; secondly, for each label set we consider the mass, i.e. element of $[0, 1]$, of the current state distribution that does not deadlock, i.e. can perform some of the labels. These two aspects form the crux of our approach and distinguish it from other approaches.

3 Applications

We now argue by some concrete application domains that the distribution view on bisimulation yields a fruitful notion.

Memoryless vs. memoryfull continuous time. First, we reconsider the motivating discussion from Section 1 revolving around the difference between continuous time represented by real-valued clocks, respectively memoryless stochastic time. For this we introduce a simple model of *stochastic automata* [10].

Definition 3. *A* stochastic automaton (SA) *is a tuple $\mathcal{S} = (\mathcal{Q}, \mathcal{C}, \mathcal{A}, \rightarrow, \kappa, F)$ where $\mathcal{Q}$ is a set of locations, $\mathcal{C}$ is a set of clocks, $\mathcal{A}$ is a set of actions, $\rightarrow \subseteq \mathcal{Q} \times \mathcal{A} \times 2^{\mathcal{C}} \times \mathcal{Q}$ is a set of edges, $\kappa : \mathcal{Q} \to 2^{\mathcal{C}}$ is a clock setting function, and F assigns to each clock its distribution over $\mathbb{R}_{\geq 0}$.*

Avoiding technical details, $\mathcal{S}$ has the following NLMP semantics $\mathbf{P}_\mathcal{S}$ with state space $S = \mathcal{Q} \times \mathbb{R}^\mathcal{C}$, assuming it is initialized in some location q_0: When a location q is entered, for each clock $c \in \kappa(q)$ a positive *value* is chosen randomly according to the distribution $F(c)$ and stored in the state space. Intuitively, the automaton idles in location q with all clock values decreasing at the same speed until some edge (q, a, X, q') becomes *enabled*, i.e. all clocks from X have value ≤ 0. After this *idling time* t, the action a is taken and the automaton enters the next location q'. If an edge is enabled on entering a location, it is taken immediately, i.e. $t = 0$. If more than one edge become enabled simultaneously, one of them is chosen non-deterministically. Its formal definition is given in [31]. We now are in the position to harvest Definition 2, to arrive at the novel bisimulation for stochastic automata.

Definition 4. *We say that locations q_1, q_2 of an SA $\mathcal{S}$ are* probabilistic bisimilar*, denoted $q_1 \sim q_2$, if $\mu_{q_1} \sim \mu_{q_2}$ in $\mathbf{P}_\mathcal{S}$ where μ_q denotes a distribution over the state space of $\mathbf{P}_\mathcal{S}$ given by the location being q, every $c \notin \kappa(q)$ being* 0*, and every $c \in \kappa(q)$ being independently set to a random value according to $F(c)$.*

This bisimulation identifies q and q' from Section 1 unlike any previous bisimulation on SA [10]. In Section 4 we discuss how to compute this bisimulation, despite being continuous-space. Recall that the model initialized by q is obtained by first translating two simple CTMC, and then applying the natural interleaving semantics, while the model, of q' is obtained by first applying the equally natural CTMC interleaving semantics prior to translation. The bisimilarity of these two models generalizes to the whole universe of CTMC and SA:

Theorem 1. *Let $SA(\mathcal{C})$ denote the stochastic automaton corresponding to a CTMC $\mathcal{C}$. For any CTMC $\mathcal{C}_1, \mathcal{C}_2$, we have*

$$SA(\mathcal{C}_1) \parallel_{SA} SA(\mathcal{C}_1) \quad \sim \quad SA(\mathcal{C}_1 \parallel_{CT} \mathcal{C}_1).$$

Here, $\parallel_{CT}$ and $\parallel_{SA}$ denotes the interleaving parallel composition of SA [11] (echoing TA parallel composition) and CTMC [33,30] (Kronecker sum of their matrix representations), respectively.

Bisimulation for partial-observation MDP (POMDP). A POMDP is a quadruple $\mathcal{M} = (S, \mathcal{A}, \delta, \mathcal{O})$ where (as in an MDP) S is a set of states, $\mathcal{A}$ is a set of actions, and $\delta : S \times \mathcal{A} \to \mathcal{D}(S)$ is a transition function. Furthermore, $\mathcal{O} \subseteq 2^S$ partitions the state space. The choice of actions is resolved by a policy yielding a Markov chain. Unlike in an MDP, such choice is not based on the knowledge of the current state, only on knowing that the current state belongs into an *observation* $o \in \mathcal{O}$. POMDPs have a wide range of applications in robotic control, automated planning, dialogue systems, medical diagnosis, and many other areas [46].

In the analysis of POMDP, the distributions over states, called *beliefs*, arise naturally and bisimulations over beliefs have already been considered [7,34].

However, to the best of our knowledge, no algorithms for computing belief bisimilation for POMDP exist. We fill this gap by our algorithm for computing distribution bisimulation for PA in Section 4. Indeed, two beliefs μ, ν in POMDP $\mathcal{M}$ are belief bisimilar in the spirit of [7] iff μ and ν are distribution bisimilar in the induced PA $D_{\mathcal{M}} = (S, \mathcal{O} \times \mathcal{A}, \longrightarrow)$ where $(s, (o, a), \mu) \in \longrightarrow$ if $s \in o$ and $\delta(s, a) = \mu$.(3)

Further applications. Probabilistic automata are especially apt for compositional modelling of *distributed systems*. The only information a component in a distributed system has about the current state of another component stems from their mutual communication. Therefore, each component can be also viewed from the outside as a partial-observation system. Thus, also in this context, distribution bisimulation is a natural concept. While $\sim$ is not a congruence w.r.t. standard parallel composition, it is apt for compositional modelling of distributed systems where only *distributed schedulers* are considered. For details, see [31,49].

Furthermore we can understand a PA as a description, in the sense of [25,39], of a representative *agent* in a large homogeneous *population*. The distribution view then naturally represents the ratios of agents being currently in the individual states and labels given to this large population of PAs correspond to global control actions [25]. For more details on applications, see [31].

4 Algorithms

In this section, we discuss computational aspects of deciding our bisimulation. Since $\sim$ is a relation over distributions over the system's state space, it is uncountably infinite even for simple finite systems, which makes it in principle intricate to decide. Fortunately, the bisimulation relation has a linear structure, and this allows us to employ methods of linear algebra to work with it effectively. Moreover, important classes of continuous-space systems can be dealt with, since their structure can be exploited. We exemplify this on a subset of deterministic stochastic automata, for which we are able to provide an algorithm to decide bisimilarity.

Finite systems – greatest fixpoints. Let us fix a PA $(S, L, \longrightarrow)$. We apply the standard approach by starting with $\mathcal{D}(S) \times \mathcal{D}(S)$ and pruning the relation until we reach the fixpoint $\sim$. In order to represent $\sim$ using linear algebra, we identify a distribution μ with a vector $(\mu(s_1), \ldots, \mu(s_{|S|})) \in \mathbb{R}^{|S|}$.

Although the space of distributions is uncountable, we construct an implicit representation of $\sim$ by a system of equations written as columns in a matrix E.

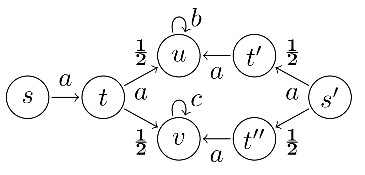
Definition 5. *A matrix E with $|S|$ rows is a* bisimulation matrix *if for some bisimulation R, for any distributions μ, ν*

$$\mu \, R \, \nu \quad \textit{iff} \quad (\mu - \nu)E = \mathbf{0}.$$

For a bisimulation matrix E, an equivalence class of μ is then the set $(\mu + \{\rho \mid \rho E = \mathbf{0}\}) \cap \mathcal{D}(S)$, the set of distributions that are equal modulo E.

(3) Note that [7] also considers rewards that can be easily added to $\sim$ and our algorithm.

Example 5. The bisimulation matrix E below encodes that several conditions must hold for two distributions μ, ν to be bisimilar. Among others, if we multiply $\mu - \nu$ with e.g. the second column, we must get 0. This translates to $(\mu(v) - \nu(v)) \cdot \mathbf{1} = 0$, i.e. $\mu(v) = \nu(v)$. Hence for bisimilar distributions, the measure of v has to be the same. This proves that $u \not\sim v$ (here we identify states and their Dirac distributions). Similarly, we can prove that $t \sim \frac{1}{2}t' + \frac{1}{2}t''$. Indeed, if we multiply the corresponding difference vector $(0, 0, 1, -\frac{1}{2}, -\frac{1}{2}, 0, 0)$ with any column of the matrix, we obtain 0.



$$\begin{array}{r} s: \\ s': \\ t: \\ t': \\ t'': \\ u: \\ v: \end{array} \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \end{pmatrix}$$

Note that the unit matrix is always a bisimulation matrix, not relating anything with anything but itself. For which bisimulations do there exist bisimulation matrices? We say a relation R over distributions is *convex* if $\mu R \nu$ and $\mu' R \nu'$ imply $\big(p\mu + (1-p)\mu'\big)\ R\ \big(p\nu + (1-p)\nu'\big)$ for any $p \in [0, 1]$.

Lemma 1. *Every convex bisimulation has a corresponding bisimulation matrix.*

Since $\sim$ is convex (see [31]), there is a bisimulation matrix corresponding to $\sim$. It is a least restrictive bisimulation matrix E (note that all bisimulation matrices with the least possible dimension have identical solution space), we call it *minimal bisimulation matrix.* We show that the necessary and sufficient condition for E to be a bisimulation matrix is *stability* with respect to transitions.

Definition 6. *For a $|S| \times |S|$ matrix P, we say that a matrix E with $|S|$ rows is P-*stable *if for every $\rho \in \mathbb{R}^{|S|}$,*

$$\rho E = \mathbf{0} \implies \rho P E = \mathbf{0} \tag{1}$$

We first briefly explain the stability in a simpler setting.

Action-deterministic systems. Let us consider PA where in each state, there is at most one transition. For each $a \in L$, we let $P_a = (p_{ij})$ denote the transition matrix such that for all i, j, if there is (unique) transition $s_i \xrightarrow{a} \mu$ we set p_{ij} to $\mu(s_j)$, otherwise to 0. Then μ evolves under a into μP_a. Denote $\mathbf{1} = (1, \dots, 1)^\top$.

Proposition 1. *In an action-deterministic PA, E containing $\mathbf{1}$ is a bisimulation matrix iff it is P_a-stable for all $a \in L$.*

To get a minimal bisimulation matrix E, we start with a single vector $\mathbf{1}$ which stands for an equation saying that the overall probability mass in bisimilar distributions is the same. Then we repetitively multiply all vectors we have by all the matrices P_a and add each resulting vector to the collection if it is linearly independent of the current collection, until there are no changes. In Example 5, the second column of E is obtained as $P_c\mathbf{1}$, the fourth one as $P_a(P_c\mathbf{1})$ and so on.

The set of all columns of E is thus given by the described iteration

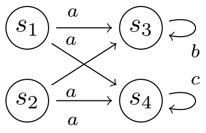
$$\{P_a \mid a \in L\}^* \mathbf{1}$$

modulo linear dependency. Since P_a have $|S|$ rows, the fixpoint is reached within $|S|$ iterations yielding $1 \leq d \leq |S|$ equations. Each class then forms an $(|S| - d)$-dimensional affine subspace intersected with the set of probability distributions $\mathcal{D}(S)$. This is also the principle idea behind the algorithm of [51] and [19].

Non-deterministic systems. In general, for transitions under A, we have to consider c_i^A non-deterministic choices in each s_i among all the outgoing transitions under some $a \in A$. We use variables w_i^j denoting the probability that j-th transition, say (s_i, a_i^j, μ_i^j), is taken by the scheduler/player[(4)] in s_i. We sum up the choices into a "non-deterministic" transition matrix P_A^W with parameters W whose ith row equals $\sum_{j=1}^{c_i^A} w_i^j \mu_i^j$. It describes where the probability mass moves from s_i under A depending on the collection W of the probabilities the player gives each choice. By $\mathcal{W}_A$ we denote the set of all such W.

A simple generalization of the approach above would be to consider $\{P_A^W \mid A \subseteq L, W \in \mathcal{W}_A\}^* \mathbf{1}$. However, firstly, the set of these matrices is uncountable whenever there are at least two transitions to choose from. Secondly, not all P_A^W may be used as the following example shows.

Example 6. In each bisimulation class in the following example, the probabilities of $s_1 + s_2$, s_3, and s_4 are constant, as can also be seen from the bisimulation matrix E, similarly to Example 5. Further, E can be obtained as $(\mathbf{1}\ P_c\mathbf{1}\ P_b\mathbf{1})$. Observe that E is $P_{\{a\}}^W$-stable for W that maximizes the probability of going into the "class" s_3 (both s_1 and s_2 go to s_3, i.e. $w_1^1 = w_2^1 = 1$); similarly for the "class" s_4.


$$\begin{tikzcd}
s_1 \arrow[r, "a"] \arrow[dr, "a"'] & s_3 \arrow[loop right, "b"] \\
s_2 \arrow[r, "a"'] \arrow[ur, "a"] & s_4 \arrow[loop right, "c"]
\end{tikzcd}$$

$$P_{\{a\}}^W = \begin{pmatrix} 0 & 0 & w_1^1 & w_2^2 \\ 0 & 0 & w_2^1 & w_2^2 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \qquad E = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

However, for $\overline{W}$ with $w_1^1 \neq w_2^1$, e.g. s_1 goes to s_3 and s_2 goes with equal probability to s_3 and s_4 ($w_1^1 = 1, w_2^1 = w_2^2 = \frac{1}{2}$), we obtain from $P_{\{a\}}^{\overline{W}} E$ a new independent vector $(0, 0.5, 0, 0)^\top$ enforcing a partition finer than $\sim$. This does not mean that Spoiler wins the game when choosing such mixed $\overline{W}$ in some μ, it only means that Duplicator needs to choose a *different* W in a bisimilar ν in order to have $\mu P_A^{\overline{W}} \sim \nu P_A^W$ for the successors.

---

(4) We use the standard notion of Spoiler-Duplicator bisimulation game (see e.g. [42]) where in $\{\mu_0, \mu_1\}$ Spoiler chooses $i \in \{0,1\}, A \subseteq L$, and $\mu_i \xrightarrow{A} \mu_i'$, Duplicator has to reply with $\mu_{1-i} \xrightarrow{A} \mu_{1-i}'$ such that $\mu_i(S_A) = \mu_{i-1}(S_A)$, and the game continues in $\{\mu_0', \mu_1'\}$. Spoiler wins iff at some point Duplicator cannot reply.

A fundamental observation is that we get the correct bisimulation when Spoiler is restricted to finitely many "extremal" choices and Duplicator is restricted for such extremal W to respond only with the very same W. $(*)$

To this end, consider $M_A^W = P_A^W E$ where E is the current matrix with each of e columns representing an equation. Intuitively, the ith row of M_A^W describes how much of s_i is moved to various classes when a step is taken. Denote the linear forms in M_A^W over W by m_{ij}. Since the players can randomize and mix choices which transition to take, the set of vectors $\{(m_{i1}(w_i^1, \dots, w_i^{c_i}), \dots, m_{ib}(w_i^1, \dots, w_i^{c_i})) \mid w_i^1, \dots, w_i^{c_i} \geq 0, \sum_{j=1}^{c_i} w_i^j = 1\}$ forms a convex polytope denoted by C_i. Each vector in C_i is thus the ith row of the matrix M_A^W where some concrete weights w_i^j are "plugged in". This way C_i describes all the possible choices in s_i and their effect on where the probability mass is moved.

Denote vertices (extremal points) of a convex polytope P by $\mathcal{E}(P)$. Then $\mathcal{E}(C_i)$ correspond to pure (non-randomizing) choices that are "extremal" w.r.t. E. Note that now if $s_j \sim s_k$ then $C_j = C_k$, or equivalently $\mathcal{E}(C_j) = \mathcal{E}(C_k)$. Indeed, for every choice in s_j there needs to be a matching choice in s_k and vice versa. However, since we consider bisimulation between generally non-Dirac distributions, we need to combine these extremal choices. For an arbitrary distribution $\mu \in \mathcal{D}(S)$, we say that a tuple $c \in \prod_{i=1}^{|S|} \mathcal{E}(C_i)$ is *extremal in* μ if $\mu \cdot c^\top$ is a vertex of the polytope $\{\mu \cdot c'^\top \mid c' \in \prod_{i=1}^{|S|} C_i\}$. Note that each extremal c corresponds to particular pure choices, denoted by $W(c)$. Unfortunately, for choices $W(c)$ of Spoiler extremal in *some* distribution, Duplicator may in *another* distribution need to make different choices. Indeed, in Example **??** the tuple corresponding to $\overline{W}$ is extremal in the Dirac distribution of state s_1. Therefore, we define $\mathcal{E}(C)$ to be the set of tuples c extremal in the uniform distribution. Interestingly, tuples extremal in the uniform distribution are (1) extremal in all distributions and (2) reflect all extremal choices, i.e. for every c extremal in some μ, there is a c' extremal in the uniform distribution such that c' is also extremal in μ and $\mu \cdot c = \mu \cdot c'$. As a result, the fundamental property $(*)$ is guaranteed.

Proposition 2. *Let E be a matrix containing* $\mathbf{1}$. *It is a bisimulation matrix iff it is $P_A^{W(c)}$-stable for all $A \subseteq L$ and $c \in \mathcal{E}(C)$.*
Theorem 2. *Algorithm 1 computes a minimal bisimulation matrix.*

The running time is exponential. We leave the question whether linear programming or other methods [32] can yield E in polynomial time open. The algorithm can easily be turned into one computing other bisimulation notions from the literature, for which there were no algorithms so far, see Section 5.

Continuous-time systems - least fixpoints. Turning our attention to continuous systems, we finally sketch an algorithm for deciding bisimulation $\sim$ over a subclass of stochastic automata, this constitutes the first algorithm to compute a bisimulation on the uncountably large semantical object.

We need to adopt two restrictions. First, we consider only *deterministic* SA, where the probability that two edges become enabled at the same time is zero (when initiated in any location). Second, to simplify the exposition, we restrict all

```
Input    : Probabilistic automaton $(S, L, \rightarrow)$
Output   : A minimal bisimulation matrix $E$
foreach $A \subseteq L$ do
    compute $P_A^W$                                   // non-deterministic transition matrix
$E \leftarrow (\mathbf{1})$
repeat
    foreach $A \subseteq L$ do
        $M_A^W \leftarrow P_A^W E$                    // polytope of all choices
        compute $\mathcal{E}(C)$ from $M_A^W$         // vertices, i.e. extremal choices
        foreach $c \in \mathcal{E}(C)$ do
            $M_A^{W(c)} \leftarrow M_A^W$ with values $W(c)$ plugged in
            $E_{new} \leftarrow$ columns of $M_A^{W(c)}$ linearly independent of columns of $E$
            $E \leftarrow (E\ E_{new})$
until $E$ does not change
```

Algorithm 1: Bisimulation on probabilistic automata

distributions occurring to exponential distributions. Notably, even for this class, our bisimulation is strictly coarser than the one induced by standard bisimulations [33,30,6] for continuous-time Markov chains. At the end of the section we discuss possibilities for extending the class of supported distributions. Both the restrictions can be effectively checked on SA.

Theorem 3. *Let $\mathcal{S} = (\mathcal{Q}, \mathcal{C}, \mathcal{A}, \rightarrow, \kappa, F)$ be a deterministic SA over exponential distributions. There is an algorithm to decide in time polynomial in $|\mathcal{S}|$ and exponential in $|\mathcal{C}|$ whether $q_1 \sim q_2$ for any locations q_1, q_2.*

The rest of the section deals with the proof. We fix $\mathcal{S} = (\mathcal{Q}, \mathcal{C}, \mathcal{A}, \rightarrow, \kappa, F)$ and $q_1, q_2 \in \mathcal{Q}$. First, we straightforwardly abstract the NLMP semantics $\mathbf{P}_{\mathcal{S}}$ over state space $S = \mathcal{Q} \times \mathbb{R}^{\mathcal{C}}$ by a NLMP $\hat{\mathbf{P}}$ over state space $\hat{S} = \mathcal{Q} \times (\mathbb{R}_{\geq 0} \cup \{-\})^{\mathcal{C}}$ where all negative values of clocks are expressed by $-$. Let ξ denote the obvious mapping of distributions $\mathcal{D}(S)$ onto $\mathcal{D}(\hat{S})$. Then ξ preserves bisimulation since two states s_1, s_2 that differ only in negative values satisfy $\xi(\tau_a(s_1)) = \xi(\tau_a(s_2))$ for all $a \in L$.

Lemma 2. *For any distributions μ, ν on S we have $\mu \sim \nu$ iff $\xi(\mu) \sim \xi(\nu)$.*

Second, similarly to an embedded Markov chain of a CTMC, we further abstract the NLMP $\hat{\mathbf{P}}$ by a *finite* deterministic PA $\bar{D} = (\bar{S}, \mathcal{A}, \longrightarrow)$ such that each state of $\bar{D}$ is a distribution over the uncountable state space $\hat{S}$.

- The set $\bar{S}$ is the set of states reachable via the transitions relation defined below from the distributions μ_{q_1}, μ_{q_2} corresponding to q_1, q_2 (see Definition 4).
- Let us fix a state $\mu \in \bar{S}$ (note that $\mu \in \mathcal{D}(\hat{S})$) and an action $a \in \mathcal{A}$ such that in the NLMP $\hat{\mathbf{P}}$ an a-transition occurs with positive probability, i.e. $\mu \xrightarrow{A_a} \nu$ for some ν and for $A_a = \{a\} \times \mathbb{R}_{\geq 0}$. Thanks to restricting to deterministic SA, $\hat{\mathbf{P}}$ is also deterministic and such a distribution ν

is uniquely defined. We set $(\mu, a, M) \in \longrightarrow$ where M is the discrete distribution that assigns probability $p_{q,f}$ to state $\nu_{q,f}$ for each $q \in \mathcal{Q}$ and $f : \mathcal{C} \to \{-, +\}$ where $p_{q,f} = \nu(\hat{S}_{q,f})$, $\nu_{q,f}$ is the conditional distribution $\nu_q(X) := \nu(X \cap \hat{S}_{q,f})/\nu(\hat{S}_{q,f})$ for any measurable $X \subseteq \hat{S}$, and $\hat{S}_{q,f} = \{(q', v) \in \hat{S} \mid q' = q, v(c) \geq 0 \text{ iff } f(c) = + \text{ for each } c \in \mathcal{C}\}$ the set of states with location q and where the sign of clock values matches f.

For exponential distributions all the reachable states $\nu \in \bar{S}$ correspond to some location q where the subset $X \subseteq \mathcal{C}$ is newly sampled, hence we obtain:

Lemma 3. *For a deterministic SA over exponential distributions, $|\bar{S}| \leq |\mathcal{Q}| \cdot 2^{|\mathcal{C}|}$.*

Instead of a greatest fixpoint computation as employed for the discrete algorithm, we take a complementary approach and prove or disprove bisimilarity by a least fixpoint procedure. We start with the initial pair of distributions (states in $\bar{D}$) which generates further requirements that we impose on the relation and try to satisfy them. We work with a *tableau*, a rooted tree where each node is either an *inner node* with a pair of discrete probability distributions over states of $\bar{D}$ as a label, a *repeated node* with a label that already appears somewhere between the node and the root, or a *failure node* denoted by $\Box$, and the children of each inner node are obtained by one *rule* from $\{\mathbf{Step}, \mathbf{Lin}\}$. A tableau not containing $\Box$ is *successful.*

Step For a node $\mu \sim \nu$ where μ and ν have *compatible timing*, we add for each label $a \in L$ one child node $\mu_a \sim \nu_a$ where μ_a and ν_a are the unique distributions such that $\mu \xrightarrow{a} \mu_a$ and $\nu \xrightarrow{a} \nu_a$. Otherwise, we add one failure node. We say that μ and ν have compatible timing if for all actions $a \in \mathcal{A}$ we have that $T_a[\mu]$ is equivalent to $T_a[\nu]$. Here $T_a[\rho]$ is a measure over $\mathbb{R}_{\geq 0}$ such that $T_a[\rho](I) := \rho(\hat{S}_{\{a\} \times I})$, i.e. the measure of states moving after time in I with action a.

Lin For a node $\mu \sim \nu$ linearly dependent on the set of remaining nodes in the tableau, we add one child (repeat) node $\mu \sim \nu$. Here, we understand each node $\mu \sim \nu$ as a vector $\mu - \nu$ in the $|S_{\mathcal{S}}|$-dimensional vector space.

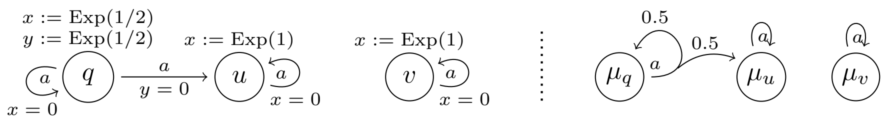
Note that compatibility of timing is easy to check. Furthermore, the set of rules is correct and complete w.r.t. bisimulation in $\hat{\mathbf{P}}$.

Lemma 4. *There is a successful tableau from $\mu \sim \nu$ iff $\mu \sim \nu$ in $\hat{\mathbf{P}}$. Moreover, the set of nodes of a successful tableau is a subset of a bisimulation.*

We get Theorem 3 since $q_1 \sim q_2$ iff $\xi(\mu_{q_1}) \sim \xi(\mu_{q_2})$ in $\hat{\mathbf{P}}$ and since, thanks to **Lin**:

Lemma 5. *There is a successful tableau from $\mu \sim \nu$ iff there is a finite successful tableau from $\mu \sim \nu$ of size polynomial in $|\bar{S}|$.*

Example 7. Let us demonstrate the rules by a simple example. Consider the following stochastic automaton $\mathcal{S}$ on the left.



Thanks to the exponential distributions, $\bar{D}$ on the right has also only three states where $\mu_q = q \otimes Exp(1/2) \otimes Exp(1/2)$ is the product of two exponential distributions with rate $1/2$, $\mu_u = u \otimes Exp(1)$, and $\mu_v = v \otimes Exp(1)$. Note that for both clocks x and y, the probability of getting to zero first is 0.5.

$$\dfrac{1 \cdot \mu_u \;\sim\; 1 \cdot \mu_v}{1 \cdot \mu_u \;\sim\; 1 \cdot \mu_v}\,\textbf{Step} \qquad\qquad \dfrac{\dfrac{\dfrac{1 \cdot \mu_q + 0 \cdot \mu_u \;\sim\; 1 \cdot \mu_v}{\frac{1}{2} \cdot \mu_q + \frac{1}{2} \cdot \mu_u \;\sim\; 1 \cdot \mu_v}\,\textbf{Step}}{\frac{1}{4} \cdot \mu_q + \frac{3}{4} \cdot \mu_u \;\sim\; 1 \cdot \mu_v}\,\textbf{Step}}{\cdots}\,\textbf{Step}$$

The finite tableau on the left is successful since it ends in a repeated node, thus it proves $u \sim v$. The infinite tableau on the right is also successful and proves $q \sim v$. When using only the rule **Step**, it is necessarily infinite as no node ever repeats. The rule **Lin** provides the means to truncate such infinite sequences. Observe that the third node in the tableau on the right above is linearly dependent on its ancestors.

Remark 1. Our approach can be turned into a complete proof system for bisimulation on models with *expolynomial* distributions [(5)]. For them, the states of the discrete transition system $\bar{D}$ can be expressed symbolically. In fact, we conjecture that the resulting semi-algorithm can be twisted to a decision algorithm for this expressive class of models. This is however out of the scope of this paper.

5 Related work and discussion

For an overview of coalgebraic work on probabilistic bisimulations we refer to a survey [47]. A considerable effort has been spent to extend this work to continuous-space systems: the solution of [15] (unfortunately not applicable to $\mathbb{R}$), the construction of [21] (described by [42] as "ingenious and intricate"), sophisticated measurable selection techniques in [18], and further approaches of [17] or [52]. In contrast to this standard setting where relations between states and their successor distributions must be handled, our work uses directly relations on distributions which simplifies the setting. The coalgebraic approach has also been applied to trace semantics of uncountable systems [35]. The topic is still very lively, e.g. in the recent [41] a different coalgebraic description of the classical probabilistic bisimulation is given.

Recently, distribution-based bisimulations have been studied. In [19], a bisimulation is defined in the context of language equivalence of Rabin's deterministic

[(5)] With density that is positive on an interval $[\ell, u)$ for $\ell \in \mathbb{N}_0$, $u \in \mathbb{N} \cup \{\infty\}$ given piecewise by expressions of the form $\sum_{i=0}^{I} \sum_{j=0}^{J} a_{ij} x^i e^{-\lambda_{ij} x}$ for $a_{ij}, \lambda_{ij} \in \mathbb{R} \cup \{\infty\}$. This class contains many important distributions such as exponential, or uniform, and enables efficient approximation of others.

probabilistic automata and also an algorithm to compute the bisimulation on them. However, only finite systems with no non-determinism are considered. The most related to our notion are the very recent independently developed [24] and [49]. However, none of them is applicable in the continuous setting and for neither of the two any algorithm has previously been given. Nevertheless, since they are close to our definition, our algorithm with only small changes can actually compute them. Although the bisimulation of [24] in a rather complex way extends [19] to the non-deterministic case reusing their notions, it can be equivalently rephrased as our Definition 2 only considering singleton sets $A \subseteq L$. Therefore, it is sufficient to only consider matrices P_A^W for singletons A in our algorithm. Apart from being a weak relation, the bisimulation of [49] differs in the definition of $\mu \xrightarrow{A} \nu$: instead of restricting to the states of the support that can perform *some* action of A, it considers those states that can perform *exactly* actions of A. Here each ith row of each transition matrix P_A^W needs to be set to zero if the set of labels from s_i is different from A.

There are also bisimulation relations over distributions defined over finite [9,29] or uncountable [8] state spaces. They, however, coincide with the classical [38] on Dirac distributions and are only directly lifted to non-Dirac distributions. Thus they fail to address the motivating correspondence problem from Section 1. Moreover, no algorithms were given. Further, weak bisimulations [23,22,16] (coarser than usual state based analogues) applied to models without internal transitions also coincide with lifting [29] of the classical bisimulation [38] while our bisimulation is coarser.

There are other bisimulations that identify more states than the classical [38] such as [48] and [4] designed to match a specific logic. Another approach to obtain coarser equivalences on probabilistic automata is via testing scenarios [50].

6 Conclusion

We have introduced a general and natural notion of a distribution-based probabilistic bisimulation, have shown its applications in different settings and have provide algorithms to compute it for finite and some classes of infinite systems. As to future work, the precise complexity of the finite case is certainly of interest. Further, the tableaux decision method opens the arena for investigating wider classes of continuous-time systems where the new bisimulation is decidable.

Acknowledgement We would like to thank Pedro D'Argenio, Filippo Bonchi, Daniel Gebler, and Matteo Mio for valuable feedback and discussions.

References

1. M. Agrawal, S. Akshay, B. Genest, and P. Thiagarajan. Approximate verification of the symbolic dynamics of Markov chains. In *LICS*, 2012.
2. R. Alur and D. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.
3. G. Behrmann, A. David, K. G. Larsen, P. Pettersson, and W. Yi. Developing UPPAAL over 15 years. *Softw., Pract. Exper.*, 41(2):133–142, 2011.

4. M. Bernardo, R. D. Nicola, and M. Loreti. Revisiting bisimilarity and its modal logic for nondeterministic and probabilistic processes. Technical Report 06, IMT Lucca, 2013.
5. M. Bravetti and P. D'Argenio. Tutte le algebre insieme: Concepts, discussions and relations of stochastic process algebras with general distributions. In *Validation of Stochastic Systems*, 2004.
6. M. Bravetti, H. Hermanns, and J.-P. Katoen. YMCA: Why Markov Chain Algebra? *Electr. Notes Theor. Comput. Sci.*, 162:107–112, 2006.
7. P. Castro, P. Panangaden, and D. Precup. Equivalence relations in fully and partially observable Markov decision processes. In *IJCAI*, 2009.
8. S. Cattani. *Trace-based Process Algebras for Real-Time Probabilistic Systems.* PhD thesis, University of Birmingham, 2005.
9. S. Crafa and F. Ranzato. A spectrum of behavioral relations over ltss on probability distributions. In *CONCUR*, 2011.
10. P. D'Argenio and J.-P. Katoen. A theory of stochastic systems, part I: Stochastic automata. *Inf. Comput.*, 203(1):1–38, 2005.
11. P. D'Argenio and J.-P. Katoen. A theory of stochastic systems, part II: Process algebra. *Inf. Comput.*, 203(1):39–74, 2005.
12. P. R. D'Argenio and C. Baier. What is the relation between CTMC and TA?, 1999. Personal communication.
13. A. David, K. Larsen, A. Legay, M. Mikucionis, D. Poulsen, J. van Vliet, and Z. Wang. Statistical model checking for networks of priced timed automata. In *FORMATS*, 2011.
14. A. David, K. Larsen, A. Legay, M. Mikucionis, and Z. Wang. Time for statistical model checking of real-time systems. In *CAV*, 2011.
15. E. de Vink and J. Rutten. Bisimulation for probabilistic transition systems: A coalgebraic approach. In *ICALP*, 1997.
16. Y. Deng and M. Hennessy. On the semantics of Markov automata. *Inf. Comput.*, 222:139–168, 2013.
17. J. Desharnais, V. Gupta, R. Jagadeesan, and P. Panangaden. Approximating labeled Markov processes. In *LICS*, 2000.
18. E.-E. Doberkat. Semi-pullbacks and bisimulations in categories of stochastic relations. In *ICALP*, 2003.
19. L. Doyen, T. Henzinger, and J.-F. Raskin. Equivalence of labeled Markov chains. *Int. J. Found. Comput. Sci.*, 19(3):549–563, 2008.
20. L. Doyen, T. Massart, and M. Shirmohammadi. Limit synchronization in Markov decision processes. *CoRR*, abs/1310.2935, 2013.
21. A. Edalat. Semi-pullbacks and bisimulation in categories of Markov processes. *Mathematical Structures in Computer Science*, 9(5):523–543, 1999.
22. C. Eisentraut, H. Hermanns, J. Krämer, A. Turrini, and L. Zhang. Deciding bisimilarities on distributions. In *QEST*, 2013.
23. C. Eisentraut, H. Hermanns, and L. Zhang. On probabilistic automata in continuous time. In *LICS*, 2010.
24. Y. Feng and L. Zhang. When equivalence and bisimulation join forces in probabilistic automata. In *FM*, 2014.
25. N. Gast and B. Gaujal. A mean field approach for optimization in discrete time. *Discrete Event Dynamic Systems*, 21(1):63–101, 2011.
26. S. Georgievska and S. Andova. Probabilistic may/must testing: retaining probabilities by restricted schedulers. *Formal Asp. Comput.*, 24(4-6):727–748, 2012.
27. M. Giry. A categorical approach to probability theory. In *Categorical aspects of topology and analysis*. Springer, 1982.

28. P. G. Harrison and B. Strulo. Spades - a process algebra for discrete event simulation. *J. Log. Comput.*, 10(1):3–42, 2000.
29. M. Hennessy. Exploring probabilistic bisimulations, part I. *Formal Asp. Comput.*, 2012.
30. H. Hermanns, U. Herzog, and V. Mertsiotakis. Stochastic process algebras - between LOTOS and Markov chains. *Computer Networks*, 30(9-10):901–924, 1998.
31. H. Hermanns, J. Krčál, and J. Křetínský. Probabilistic bisimulation: Naturally on distributions. *CoRR*, abs/1404.5084, 2014.
32. H. Hermanns and A. Turrini. Deciding probabilistic automata weak bisimulation in polynomial time. In *FSTTCS*, 2012.
33. J. Hillston. *A Compositional Approach to Performance Modelling.* Cambridge University Press, New York, NY, USA, 1996.
34. D. Jansen, F. Nielson, and L. Zhang. Belief bisimulation for hidden Markov models - logical characterisation and decision algorithm. In *NASA Formal Methods*, 2012.
35. H. Kerstan and B. König. Coalgebraic trace semantics for probabilistic transition systems based on measure theory. In *CONCUR*, 2012.
36. V. Korthikanti, M. Viswanathan, G. Agha, and Y. Kwon. Reasoning about MDPs as transformers of probability distributions. In *QEST*, 2010.
37. M. Z. Kwiatkowska, G. Norman, and D. Parker. Prism 4.0: Verification of probabilistic real-time systems. In *CAV*, vol. 6806 of *Lecture Notes in Computer Science.* Springer, 2011.
38. K. Larsen and A. Skou. Bisimulation through probabilistic testing. In *POPL*, 1989.
39. R. May et al. Biological populations with nonoverlapping generations: stable points, stable cycles, and chaos. *Science*, 186(4164):645–647, 1974.
40. R. Milner. *Communication and concurrency.* PHI Series in computer science. Prentice Hall, 1989.
41. M. Mio. Upper-expectation bisimilarity and lukasiewicz μ-calculus. In *FoSSaCS*, 2014.
42. D. Sangiorgi and J. Rutten. *Advanced Topics in Bisimulation and Coinduction.* Cambridge University Press, New York, NY, USA, 1st edition, 2011.
43. R. Segala. *Modeling and Verification of Randomized Distributed Real-time Systems.* PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 1995.
44. R. Segala and N. Lynch. Probabilistic simulations for probabilistic processes. In *CONCUR*, 1994.
45. R. Segala and N. A. Lynch. Probabilistic simulations for probabilistic processes. In *CONCUR*, vol. 836 of *Lecture Notes in Computer Science.* Springer, 1994.
46. G. Shani, J. Pineau, and R. Kaplow. A survey of point-based pomdp solvers. *AAMAS*, 27(1):1–51, 2013.
47. A. Sokolova. Probabilistic systems coalgebraically: A survey. *Theor. Comput. Sci.*, 412(38):5095–5110, 2011.
48. L. Song, L. Zhang, and J. Godskesen. Bisimulations meet PCTL equivalences for probabilistic automata. In *CONCUR*, 2011.
49. L. Song, L. Zhang, and J. C. Godskesen. Late weak bisimulation for Markov automata. *CoRR*, abs/1202.4116, 2012.
50. M. Stoelinga and F. Vaandrager. A testing scenario for probabilistic automata. In *ICALP*, 2003.
51. W. Tzeng. A polynomial-time algorithm for the equivalence of probabilistic automata. *SIAM J. Comput.*, 21(2):216–227, 1992.
52. N. Wolovick. *Continuous probability and nondeterminism in labeled transaction systems.* PhD thesis, Universidad Nacional de Córdoba, 2012.

Verification of Markov Decision Processes using Learning Algorithms[⋆]

Tomáš Brázdil[1], Krishnendu Chatterjee[2], Martin Chmelík[2], Vojtěch Forejt[3], Jan Křetínský[2], Marta Kwiatkowska[3], David Parker[4], and Mateusz Ujma[3]

[1] Masaryk University, Brno, Czech Republic [2] IST Austria
[3] University of Oxford, UK [4] University of Birmingham, UK

Abstract. We present a general framework for applying machine-learning algorithms to the verification of Markov decision processes (MDPs). The primary goal of these techniques is to improve performance by avoiding an exhaustive exploration of the state space. Our framework focuses on probabilistic reachability, which is a core property for verification, and is illustrated through two distinct instantiations. The first assumes that full knowledge of the MDP is available, and performs a heuristic-driven partial exploration of the model, yielding precise lower and upper bounds on the required probability. The second tackles the case where we may only sample the MDP, and yields probabilistic guarantees, again in terms of both the lower and upper bounds, which provides efficient stopping criteria for the approximation. The latter is the first extension of statistical model checking for unbounded properties in MDPs. In contrast with other related techniques, our approach is not restricted to time-bounded (finite-horizon) or discounted properties, nor does it assume any particular properties of the MDP. We also show how our methods extend to LTL objectives. We present experimental results showing the performance of our framework on several examples.

1 Introduction

Markov decision processes (MDPs) are a widely used model for the formal verification of systems that exhibit stochastic behaviour. This may arise due to the possibility of failures (e.g. of physical system components), unpredictable events (e.g. messages sent across a lossy medium), or uncertainty about the environment (e.g. unreliable sensors in a robot). It may also stem from the explicit use of randomisation, such as probabilistic routing in gossip protocols or random back-off in wireless communication protocols.

Verification of MDPs against temporal logics such as PCTL and LTL typically reduces to the computation of optimal (minimum or maximum) reachability probabilities, either on the MDP itself or its product with some deterministic ω-automaton. Optimal reachability probabilities (and a corresponding optimal strategy for the MDP) can be computed in polynomial time through a reduction to linear programming, although in

[⋆] This research was funded in part by the European Research Council (ERC) under grant agreement 267989 (QUAREM), 246967 (VERIWARE) and 279307 (Graph Games), by the EU FP7 project HIERATIC, by the Austrian Science Fund (FWF) projects S11402-N23 (RiSE), S11407-N23 (RiSE) and P23499-N23, by the Czech Science Foundation grant No P202/12/P612, by EPSRC project EP/K038575/1 and by the Microsoft faculty fellows award.

practice verification tools often use dynamic programming techniques, such as value iteration which approximates the values up to some pre-specified convergence criterion.

The efficiency or feasibility of verification is often limited by excessive time or space requirements, caused by the need to store a full model in memory. Common approaches to tackling this include: symbolic model checking, which uses efficient data structures to construct and manipulate a compact representation of the model; abstraction refinement, which constructs a sequence of increasingly precise approximations, bypassing construction of the full model using decision procedures such as SAT or SMT; and statistical model checking [37,19], which uses Monte Carlo simulation to generate approximate results of verification that hold with high probability.

In this paper, we explore the opportunities offered by learning-based methods, as used in fields such as planning or reinforcement learning [36]. In particular, we focus on algorithms that explore an MDP by generating trajectories through it and, whilst doing so, produce increasingly precise approximations for some property of interest (in this case, reachability probabilities). The approximate values, along with other information, are used as heuristics to guide the model exploration so as to minimise the solution time and the portion of the model that needs to be considered.

We present a general framework for applying such algorithms to the verification of MDPs. Then, we consider two distinct instantiations that operate under different assumptions concerning the availability of knowledge about the MDP, and produce different classes of results. We distinguish between *complete information*, where full knowledge of the MDP is available (but not necessarily generated and stored), and *limited information*, where (in simple terms) we can only sample trajectories of the MDP.

The first algorithm assumes complete information and is based on *real-time dynamic programming* (RTDP) [3]. In its basic form, this only generates approximations in the form of lower bounds (on maximum reachability probabilities). While this may suffice in some scenarios (e.g. planning), in the context of verification we typically require more precise guarantees. So we consider bounded RTDP (BRTDP) [30], which supplements this with an additional upper bound. The second algorithm assumes limited information and is based on *delayed Q-learning* (DQL) [35]. Again, we produce both lower and upper bounds but, in contrast to BRTDP, where these are guaranteed to be correct, DQL offers probably approximately correct (PAC) results, i.e., there is a non-zero probability that the bounds are incorrect.

Typically, MDP solution methods based on learning or heuristics make assumptions about the structure of the model. For example, the presence of end components [15] (subsets of states where it is possible to remain indefinitely with probability 1) can result in convergence to incorrect values. Our techniques are applicable to arbitrary MDPs. We first handle the case of MDPs that contain no end components (except for trivial designated goal or sink states). Then, we adapt this to the general case by means of *on-the-fly* detection of end components, which is one of the main technical contributions of the paper. We also show how our techniques extend to LTL objectives and thus also to minimum reachability probabilities.

Our DQL-based method, which yields PAC results, can be seen as an instance of statistical model checking [37,19], a technique that has received considerable attention. Until recently, most work in this area focused on purely probabilistic models, without nondeterminism, but several approaches have now been presented for statistical model checking of nondeterministic models [13,14,27,4,28,18,29]. However, these methods all consider either time-bounded properties or use discounting to ensure convergence (see below for a summary). The techniques in this paper are the first for statistical model checking of unbounded properties on MDPs.

We have implemented our framework within the PRISM tool [25]. This paper concludes with experimental results for an implementation of our BRTDP-based approach that demonstrate considerable speed-ups over the fastest methods in PRISM.

Detailed proofs omitted due to lack of space are available in [7].

1.1 Related Work

In fields such as planning and artificial intelligence, many learning-based and heuristic-driven solution methods for MDPs have been developed. In the *complete information* setting, examples include RTDP [3] and BRTDP [30], as discussed above, which generate lower and lower/upper bounds on values, respectively. Most algorithms make certain assumptions in order to ensure convergence, for example through the use of a discount factor or by restricting to so-called Stochastic Shortest Path (SSP) problems, whereas we target arbitrary MDPs without discounting. More recently, an approach called FRET [24] was proposed for a generalisation of SSP, but this gives only a one-sided (lower) bound. We are not aware of any attempts to apply or adapt such methods in the context of probabilistic verification. A related paper is [1], which applies heuristic search methods to MDPs, but for generating probabilistic counterexamples.

As mentioned above, in the *limited information* setting, our algorithm based on delayed Q-learning (DQL) yields PAC results, similar to those obtained from *statistical model checking* [37,19,34]. This is an active area of research with a variety of tools [21,8,6,5]. In contrast with our work, most techniques focus on time-bounded properties, e.g., using bounded LTL, rather than *unbounded* properties. Several approaches have been proposed to transform checking of unbounded properties into testing of bounded properties, for example, [38,17,33,32]. However, these focus on purely probabilistic models, without nondeterminism, and do not apply to MDPs. In [4], unbounded properties are analysed for MDPs with spurious nondeterminism, where the way it is resolved does not affect the desired property.

More generally, the development of statistical model checking techniques for probabilistic models with *nondeterminism*, such as MDPs, is an important topic, treated in several recent papers. One approach is to give the nondeterminism a probabilistic semantics, e.g., using a uniform distribution instead, as for timed automata in [13,14,27]. Others [28,18], like this paper, aim to quantify over all strategies and produce an ε-optimal strategy. The work in [28] and [18] deals with the problem in the setting of discounted (and for the purposes of approximation thus bounded) or bounded properties, respectively. In the latter work, candidates for optimal schedulers are generated and gradually improved, but "at any given point we cannot quantify how close to optimal the candidate scheduler is" and "the algorithm does not estimate the maximum probability of the property" (cited from [29]). Further, [29] considers compact representation of schedulers, but again focuses only on (time) bounded properties.

Since statistical model checking is simulation-based, one of the most important difficulties is the analysis of *rare events*. This issue is, of course, also relevant for our approach; see the section on experimental results. Rare events have been addressed using methods such as importance sampling [17,20] and importance splitting [22].

End components in MDPs can be collapsed either for algorithmic correctness [15] or efficiency [11] (where only lower bounds on maximum reachability probabilities are considered). Asymptotically efficient ways to detect them are given in [10,9].

2 Basics about MDPs and Learning Algorithms

We begin with basic background material on MDPs and some fundamental definitions for our learning framework. We use $\mathbb{N}$, $\mathbb{Q}$, and $\mathbb{R}$ to denote the sets of all non-negative integers, rational numbers and real numbers respectively. $\mathit{Dist}(X)$ is the set of all rational probability distributions over a finite or countable set X, i.e., the functions $f : X \to [0, 1] \cap \mathbb{Q}$ such that $\sum_{x\in X} f(x) = 1$, and $\mathit{supp}(f)$ denotes the *support* of f.

2.1 Markov Decision Processes

We work with *Markov decision processes* (MDPs), a widely used model to capture both nondeterminism (e.g., for control or concurrency) and probability.

Definition 1. *An* MDP *is a tuple* $\mathsf{M} = \langle S, \overline{s}, A, E, \Delta\rangle$*, where* S *is a finite set of* states*,* $\overline{s} \in S$ *is an* initial state*,* A *is a finite set of* actions*,* $E : S \to 2^A$ *assigns non-empty sets of* enabled *actions to all states, and* $\Delta : S\times A \to \mathit{Dist}(S)$ *is a (partial)* probabilistic transition function *defined for all* s *and* a *where* $a \in E(s)$*.*

Remark 1. For simplicity of presentation we assume w.l.o.g. that, for every action $a \in A$, there is at most one state s such that $a \in E(s)$, i.e., $E(s) \cap E(s') = \emptyset$ for $s \neq s'$. If there *are* states s, s' such that $a \in E(s) \cap E(s')$, we can always rename the actions as $(s, a) \in E(s)$, and $(s', a) \in E(s')$, so that the MDP satisfies our assumption.

An *infinite path* of an MDP M is an infinite sequence $\omega = s_0a_0s_1a_1 \ldots$ such that $a_i \in E(s_i)$ and $\Delta(s_i, a_i)(s_{i+1}) > 0$ for every $i \in \mathbb{N}$. A *finite path* is a finite prefix of an infinite path ending in a state. We use $\mathit{last}(\omega)$ to denote the last state of a finite path ω. We denote by IPath (resp. FPath) the set of all infinite (resp. finite) paths, and by IPath_s (resp. FPath_s) the set of infinite (resp. finite) paths starting in a state s.

A state s is *terminal* if all actions $a \in E(s)$ satisfy $\Delta(s, a)(s) = 1$. An *end component* (EC) of M is a pair (S', A') where $S' \subseteq S$ and $A' \subseteq \bigcup_{s\in S'} E(s)$ such that: (1) if $\Delta(s, a)(s') > 0$ for some $s \in S'$ and $a \in A'$, then $s' \in S'$; and (2) for all $s, s' \in S'$ there is a path $\omega = s_0a_0 \ldots s_n$ such that $s_0 = s$, $s_n = s'$ and for all $0 \leq i < n$ we have $a_i \in A'$. A *maximal end component* (MEC) is an EC that is maximal with respect to the point-wise subset ordering.

Strategies. A *strategy* of MDP M is a function $\sigma : \mathit{FPath} \to \mathit{Dist}(A)$ satisfying $\mathit{supp}(\sigma(\omega)) \subseteq E(\mathit{last}(\omega))$ for every $\omega \in \mathit{FPath}$. Intuitively, the strategy resolves the choices of actions in each finite path by choosing (possibly at random) an action enabled in the last state of the path. We write Σ_{M} for the set of all strategies in M. In standard fashion [23], a strategy σ induces, for any initial state s, a probability measure $\mathit{Pr}^{\sigma}_{\mathsf{M},s}$ over IPath_s. A strategy σ is *memoryless* if $\sigma(\omega)$ depends only on $\mathit{last}(\omega)$.

Objectives and values. Given a set $F \subseteq S$ of target states, *bounded reachability* for step k, denoted by $\Diamond^{\leq k} F$, refers to the set of all infinite paths that reach a state in F within k steps, and *unbounded reachability*, denoted by $\Diamond F$, refers to the set of all infinite paths that reach a state in F. Note that $\Diamond F = \bigcup_{k \geq 0} \Diamond^{\leq k} F$. We consider the *reachability probability* $Pr^{\sigma}_{\mathsf{M},s}(\Diamond F)$, and strategies that maximise this probability. We denote by $V(s)$ the *value* in s, defined by $\sup_{\sigma \in \Sigma_{\mathsf{M}}} Pr^{\sigma}_{\mathsf{M},s}(\Diamond F)$. Given $\varepsilon \geq 0$, we say that a strategy σ is *ε-optimal* in s if $Pr^{\sigma}_{\mathsf{M},s}(\Diamond F) + \varepsilon \geq V(s)$, and we call a 0-optimal strategy *optimal*. It is known [31] that, for every MDP and set F, there is a memoryless optimal strategy for $\Diamond F$. We are interested in strategies that approximate the value function, i.e., ε-optimal strategies for some $\varepsilon > 0$.

2.2 Learning Algorithms for MDPs

In this paper, we study a class of learning-based algorithms that stochastically approximate the value function of an MDP. Let us fix, for this section, an MDP $\mathsf{M} = \langle S, \overline{s}, A, E, \Delta \rangle$ and target states $F \subseteq S$. We denote by $V : S \times A \to [0,1]$ the *value function* for state-action pairs of M, defined for all (s,a) where $s \in S$ and $a \in E(s)$:

$$V(s,a) := \sum\nolimits_{s' \in S} \Delta(s,a)(s') \cdot V(s').$$

Intuitively, $V(s,a)$ is the value in s assuming that the first action performed is a. A *learning algorithm* $\mathcal{A}$ simulates executions of M, and iteratively updates upper and lower approximations $U : S \times A \to [0,1]$ and $L : S \times A \to [0,1]$, respectively, of the value function $V : S \times A \to [0,1]$.

The functions U and L are initialised to appropriate values so that $L(s,a) \leq V(s,a) \leq U(s,a)$ for all $s \in S$ and $a \in A$. During the computation of $\mathcal{A}$, simulated executions start in the initial state $\overline{s}$ and move from state to state according to choices made by the algorithm. The values of $U(s,a)$ and $L(s,a)$ are updated for the states s visited by the simulated execution. Since $\max_{a \in E(s)} U(s,a)$ and $\max_{a \in E(s)} L(s,a)$ represent upper and lower bound on $V(s)$, a learning algorithm $\mathcal{A}$ terminates when $\max_{a \in E(\overline{s})} U(\overline{s},a) - \max_{a \in E(\overline{s})} L(\overline{s},a) < \varepsilon$ where the *precision* $\varepsilon > 0$ is given to the algorithm as an argument. Note that, because U and L are possibly updated based on the simulations, the computation of the learning algorithm may be randomised and even give incorrect results with some probability.

Definition 2. *Denote by $\mathcal{A}(\varepsilon)$ the instance of learning algorithm $\mathcal{A}$ with precision ε. We say that $\mathcal{A}$* converges surely (resp. almost surely) *if, for every $\varepsilon > 0$, the computation of $\mathcal{A}(\varepsilon)$ surely (resp. almost surely) terminates, and $L(\overline{s},a) \leq V(\overline{s},a) \leq U(\overline{s},a)$ holds upon termination.*

In some cases, almost-sure convergence cannot be guaranteed, so we demand that the computation terminates correctly with sufficiently high probability. In such cases, we assume the algorithm is also given a *confidence* $\delta > 0$ as an argument.

Definition 3. *Denote by $\mathcal{A}(\varepsilon, \delta)$ the instance of learning algorithm $\mathcal{A}$ with precision ε and confidence δ. We say that $\mathcal{A}$ is* probably approximately correct (PAC) *if, for every $\varepsilon > 0$ and every $\delta > 0$, with probability at least $1 - \delta$, the computation of $\mathcal{A}(\varepsilon, \delta)$ terminates with $L(\overline{s},a) \leq V(\overline{s},a) \leq U(\overline{s},a)$.*

The function U defines a memoryless strategy σ_U which in every state s chooses all actions a maximising the value $U(s,a)$ over $E(s)$ uniformly at random. The strategy σ_U is used in some of the algorithms and also contributes to the output.

Remark 2. If the value function is defined as the infimum over strategies (as in [30]), then the strategy chooses actions to minimise the lower value. Since we consider the dual case of supremum over strategies, the choice of σ_U is to maximise the upper value.

We also need to specify what knowledge about the MDP M is available to the learning algorithm. We distinguish the following two distinct cases.

Definition 4. *A learning algorithm has* limited information *about* M *if it knows only the initial state* $\overline{s}$*, a number* $K \geq |S|$*, a number* $E_m \geq \max_{s \in S} |E(s)|$*, a number* $0 < q \leq p_{\min}$*, where* $p_{\min} = \min\{\Delta(s,a)(s') \mid s \in S, a \in E(s), s' \in \mathit{supp}(\Delta(s,a))\}$*, and the function* E *(more precisely, given a state* s*, the learning procedure can ask an oracle for* $E(s)$*). We assume that the algorithm may simulate an execution of* M *starting with* $\overline{s}$ *and choosing enabled actions in individual steps.*

Definition 5. *A learning algorithm has* complete information *about* M *if it knows the complete MDP* M.

Note that the MDPs we consider are "fully observable", so even in the limited information case strategies can make decisions based on the precise state of the system.

3 MDPs without End Components

We first present algorithms for MDPs *without* ECs, which considerably simplifies the adaptation of BRTDP and DQL to unbounded reachability objectives. Later, in Section 4, we extend our methods to deal with arbitrary MDPs (*with* ECs). Let us fix an MDP $\mathsf{M} = \langle S, \overline{s}, A, E, \Delta \rangle$ and a target set F. Formally, we assume the following.

Assumption-EC. MDP M has no ECs, except two trivial ones containing distinguished terminal states 1 and 0, respectively, with $F = \{1\}$, $V(1) = 1$ and $V(0) = 0$.

3.1 Our framework

We start by formalising a general framework for learning algorithms, as outlined in the previous section. We then instantiate this and obtain two learning algorithms: BRTDP and DQL. Our framework is presented as Algorithm 1, and works as follows. Recall that functions U and L store the current upper and lower bounds on the value function V. Each iteration of the outer loop is divided into two phases: EXPLORE and UPDATE. In the EXPLORE phase (lines 5 - 10), the algorithm samples a finite path ω in M from $\overline{s}$ to a state in $\{1, 0\}$ by always randomly choosing one of the enabled actions that maximises the U value, and sampling the successor state using the probabilistic transition function. In the UPDATE phase (lines 11 - 16), the algorithm updates U and L on the state-action pairs along the path in a backward manner. Here, the function *pop* pops and returns the last letter of the given sequence.

Algorithm 1 Learning algorithm (for MDPs with no ECs)

```
1: Inputs: An EC-free MDP M
2: $U(\cdot,\cdot) \leftarrow 1, L(\cdot,\cdot) \leftarrow 0$
3: $L(1,\cdot) \leftarrow 1, U(0,\cdot) \leftarrow 0$                                  ▷ INITIALISE
4: repeat
5:     $\omega \leftarrow \overline{s}$   /* EXPLORE phase */
6:     repeat
7:         $a \leftarrow$ sampled uniformly from $\arg\max_{a \in E(last(\omega))} U(last(\omega), a)$
8:         $s \leftarrow$ sampled according to $\Delta(last(\omega), a)$          ▷ GETSUCC($\omega, a$)
9:         $\omega \leftarrow \omega\, a\, s$
10:    until $s \in \{1, 0\}$                                    ▷ TERMINATEPATH($\omega$)
11:    repeat  /* UPDATE phase */
12:        $s' \leftarrow pop(\omega)$
13:        $a \leftarrow pop(\omega)$
14:        $s \leftarrow last(\omega)$
15:        UPDATE($(s, a), s'$)
16:    until $\omega = \overline{s}$
17: until $\max_{a \in E(\overline{s})} U(\overline{s}, a) - \max_{a \in E(\overline{s})} L(\overline{s}, a) < \varepsilon$          ▷ TERMINATE
```

3.2 Instantiations: BRTDP and DQL

Our two algorithm instantiations, BRTDP and DQL, differ in the definition of UPDATE.

Unbounded reachability with BRTDP. We obtain BRTDP by instantiating UPDATE with Algorithm 2, which requires complete information about the MDP. Intuitively, UPDATE computes new values of $U(s, a)$ and $L(s, a)$ by taking the weighted average of the corresponding U and L values, respectively, over all successors of s via action a. Formally, denote $U(s) = \max_{a \in E(s)} U(s, a)$ and $L(s) = \max_{a \in E(s)} L(s, a)$.

Algorithm 2 BRTDP instantiation of Algorithm 1

```
1: procedure UPDATE($(s, a), \cdot$)
2:     $U(s, a) := \sum_{s' \in S} \Delta(s, a)(s') U(s')$
3:     $L(s, a) := \sum_{s' \in S} \Delta(s, a)(s') L(s')$
```

The following theorem says that BRTDP satisfies the conditions of Definition 2 and never returns incorrect results.

Theorem 1. *The algorithm BRTDP converges almost surely under Assumption-EC.*

Remark 3. Note that, in the EXPLORE phase, an action maximising the value of U is chosen and the successor is sampled according to the probabilistic transition function of M. However, we can consider various modifications. Actions and successors may be chosen in different ways (e.g., for GETSUCC), for instance, uniformly at random, in a round-robin fashion, or assigning various probabilities (bounded from below by some fixed $p > 0$) to all possibilities in any biased way. In order to guarantee almost-sure convergence, some conditions have to be satisfied. Intuitively we require, that the state-action pairs used by ε-optimal strategies have to be chosen enough times. If this condition is satisfied then the almost-sure convergence is preserved and the practical running times may significantly improve. For details, see Section 5.

Remark 4. The previous BRTDP algorithm is only applicable if the transition probabilities are known. However, if complete information is not known, but $\Delta(s,a)$ can be repeatedly sampled for any s and a, then a variant of BRTDP can be shown to be probably approximately correct.

Unbounded reachability with DQL. Often, complete information about the MDP is unavailable, repeated sampling is not possible, and we have to deal with only limited information about M (see Definition 4). For this scenario, we use DQL, which can be obtained by instantiating UPDATE with Algorithm 3.

Algorithm 3 DQL (delay m, estimator precision $\bar{\varepsilon}$) instantiation of Algorithm 1

```
 1: procedure UPDATE($(s,a),s'$)
 2:     if $c(s,a)=m$ and LEARN$(s,a)$ then
 3:         if $accum^U_m(s,a)/m < U(s,a)-2\bar{\varepsilon}$ then
 4:             $U(s,a) \leftarrow accum^U_m(s,a)/m + \bar{\varepsilon}$
 5:             $accum^U_m(s,a)=0$
 6:         if $accum^L_m(s,a)/m > L(s,a)+2\bar{\varepsilon}$ then
 7:             $L(s,a) \leftarrow accum^L_m(s,a)/m - \bar{\varepsilon}$
 8:             $accum^L_m(s,a)=0$
 9:         $c(s,a)=0$
10:     else
11:         $accum^U_m(s,a) \leftarrow accum^U_m(s,a)+U(s')$
12:         $accum^L_m(s,a) \leftarrow accum^L_m(s,a)+L(s')$
13:         $c(s,a) \leftarrow c(s,a)+1$
```

Macro LEARN(s,a) is true in the kth call of UPDATE$((s,a),\cdot)$ if, since the $(k-2m)$th call of UPDATE$((s,a),\cdot)$, line 4 was not executed in any call of UPDATE$(\cdot,\cdot)$.

The main idea behind DQL is as follows. As the probabilistic transition function is not known, we cannot update $U(s,a)$ and $L(s,a)$ with the actual values $\sum_{s'\in S}\Delta(s,a)(s')U(s')$ and $\sum_{s'\in S}\Delta(s,a)(s')L(s')$, respectively. However, we can instead use simulations executed in the EXPLORE phase of Algorithm 1 to estimate these values. Namely, we use $accum^U_m(s,a)/m$ to estimate $\sum_{s'\in S}\Delta(s,a)(s')U(s')$ where $accum^U_m(s,a)$ is the sum of the U values of the last m immediate successors of (s,a) seen during the EXPLORE phase. Note that the delay m must be chosen large enough for the estimates to be sufficiently close, i.e., $\bar{\varepsilon}$-close, to the real values.

So, in addition to $U(s,a)$ and $L(s,a)$, the algorithm uses new variables $accum^U_m(s,a)$ and $accum^L_m(s,a)$ to accumulate $U(s,a)$ and $L(s,a)$ values, respectively, and a counter $c(s,a)$ recording the number of invocations of a in s since the last update (all these variables are initialised to 0 at the beginning of computation). Assume that a has been invoked in s during the EXPLORE phase of Algorithm 1, which means that UPDATE$((s,a),s')$ is eventually called in the UPDATE phase of Algorithm 1 with the corresponding successor s' of (s,a). If $c(s,a)=m$ at that time, a has been invoked in s precisely m times since the last update concerning (s,a) and the procedure UPDATE$((s,a),s')$ updates $U(s,a)$ with $accum^U_m(s,a)/m$ plus an appropriate constant $\bar{\varepsilon}$ (unless LEARN is false). Here, the purpose of adding $\bar{\varepsilon}$ is to make $U(s,a)$ stay above the real value $V(s,a)$ with high probability. If $c(s,a)<m$, then

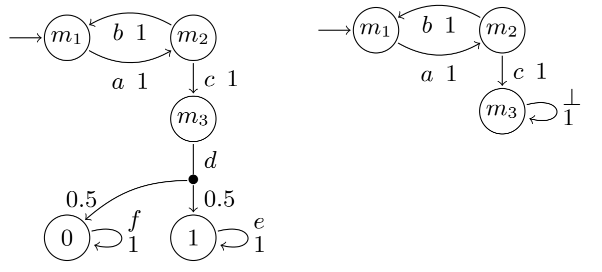


Fig. 1. MDP M with an EC (left), MDP $\mathsf{M}^{\{m_1,m_2\}}$ constructed from M in on-the-fly BRTDP (centre), and MDP M' obtained from M by collapsing $\mathcal{C} = (\{m_1, m_2\}, \{a, b\})$ (right).

UPDATE$((s, a), s')$ simply accumulates $U(s')$ into $accum^U_m(s, a)$ and increases the counter $c(s, a)$. The $L(s, a)$ values are estimated by $accum^L_m(s, a)/m$ in a similar way, just subtracting $\bar{\varepsilon}$ from $accum^L_m(s, a)$. The procedure requires m and $\bar{\varepsilon}$ as inputs, and they are chosen depending on ε and δ; more precisely, we choose $\bar{\varepsilon} = \frac{\varepsilon\cdot(p_{\min}/E_m)^{|S|}}{12|S|}$ and $m = \frac{\ln(6|S||A|(1+\frac{|S||A|}{\bar{\varepsilon}})/\delta)}{2\bar{\varepsilon}^2}$ and establish that DQL is probably approximately correct. The parameters m and $\bar{\varepsilon}$ can be conservatively approximated using only the limited information about the MDP (i.e. using K, E_m and q). Even though the algorithm has limited information about M, we still establish the following theorem.

Theorem 2. *DQL is probably approximately correct under Assumption-EC.*

Bounded reachability. Algorithm 1 can be trivially adapted to handle bounded reachability properties by preprocessing the input MDP in standard fashion. Namely, every state is equipped with a bounded counter with values ranging from 0 to k where k is the step bound, the current value denoting the number of steps taken so far. All target states remain targets for all counter values, and every non-target state with counter value k becomes rejecting. Then, to determine the k-step reachability in the original MDP, we compute the (unbounded) reachability in the new MDP. Although this means that the number of states is multiplied by $k + 1$, in practice the size of the explored part of the model can be small.

4 Unrestricted MDPs

We first illustrate with an example that the algorithms BRTDP and DQL as presented in Section 3 may not converge when there are ECs in the MDP.

Example 1. Consider the MDP M in Fig. 1 (left) with EC $(\{m_1, m_2\}, \{a, b\})$. The values in states m_1, m_2 are $V(m_1) = V(m_2) = 0.5$ but the upper bounds are $U(m_1) = U(m_2) = 1$ for every iteration. This is because $U(m_1, a) = U(m_2, b) = 1$ and both algorithms greedily choose the action with the highest upper bound. Thus, in every iteration t of the algorithm, the error for the initial state m_1 is $U(m_1) - V(m_1) = \frac{1}{2}$ and the algorithm does not converge. In general, any state in an EC has upper bound 1 since, by definition, there are actions that guarantee the next state is in the EC, i.e., is a state with upper bound 1. This argument holds even for standard value iteration with values initialised to 1.

One way of dealing with general MDPs is to preprocess them to identify all MECs [10,9] and "collapse" them into single states (see e.g. [15,11]). These algorithms require that the graph model is known and explore the whole state space, but this may not be possible either due to limited information (see Definition 4) or because the model is too large. Hence, we propose a modification to the algorithms from the previous sections that allows us to deal with ECs "on-the-fly". We first describe the collapsing of a set of states and then present a crucial lemma that allows us to identify ECs to collapse.

Collapsing states. In the following, we say that an MDP $\mathsf{M}' = \langle S', \overline{s}', A', E', \Delta' \rangle$ is obtained from $\mathsf{M} = \langle S, \overline{s}, A, E, \Delta \rangle$ by *collapsing* a tuple (R, B), where $R \subseteq S$ and $B \subseteq A$ with $B \subseteq \bigcup_{s \in R} E(s)$ if:

- $S' = (S \setminus R) \cup \{s_{(R,B)}\}$,
- $\overline{s}'$ is either $s_{(R,B)}$ or $\overline{s}$, depending on whether $\overline{s} \in R$ or not,
- $A' = A \setminus B$,
- $E'(s) = E(s)$, for $s \in S \setminus R$; $\quad E'(s_{(R,B)}) = \bigcup_{s \in R} E(s) \setminus B$,
- Δ' is defined for all $s \in S'$ and $a \in E'(s)$ by:
  - $\Delta'(s,a)(s') = \Delta(s,a)(s')$ for $s, s' \neq s_{(R,B)}$,
  - $\Delta'(s,a)(s_{(R,B)}) = \sum_{s' \in R} \Delta(s,a)(s')$ for $s \neq s_{(R,B)}$,
  - $\Delta'(s_{(R,B)},a)(s') = \Delta(s,a)(s')$ for $s' \neq s_{(R,B)}$ and s the unique state with $a \in E(s)$ (see Remark 1),
  - $\Delta'(s_{(R,B)},a)(s_{(R,B)}) = \sum_{s' \in R} \Delta(s,a)(s')$ where s is the unique state with $a \in E(s)$.

We denote the above transformation, which creates M' from M, as the COLLAPSE function, i.e., COLLAPSE(R, B). As a special case, given a state s and a terminal state $s' \in \{0, 1\}$, we use MAKETERMINAL(s, s') as shorthand for COLLAPSE$(\{s, s'\}, E(s))$, where the new state is renamed to s'. Intuitively, after MAKETERMINAL(s, s'), every transition previously leading to state s will now lead to the terminal state s'.

For practical purposes, it is important to note that the collapsing does not need to be implemented explicitly, but can be done by keeping a separate data structure which stores information about the collapsed states.

Identifying ECs from simulations. Our modifications will identify ECs "on-the-fly" through simulations that get stuck in them. The next lemma establishes the identification principle. To this end, for a path ω, let us denote by $Appear(\omega, i)$ the tuple (S_i, A_i) of M such that $s \in S_i$ and $a \in A_i(s)$ if and only if (s, a) occurs in ω more than i times.

Lemma 1. *Let $c = \exp\left(-\left(p_{\min}/E_m\right)^{\kappa} / \kappa\right)$, where $\kappa = KE_m + 1$, and let $i \geq \kappa$. Assume that the* EXPLORE *phase in Algorithm 1 terminates with probability less than* 1. *Then, provided the* EXPLORE *phase does not terminate within $3i^3$ iterations, the conditional probability that $Appear(\omega, i)$ is an EC is at least $1 - 2c^i i^3 \cdot (p_{\min}/E_m)^{-\kappa}$.*

The above lemma allows us to modify the EXPLORE phase of Algorithm 1 in such a way that simulations will be used to identify ECs. The ECs discovered will subsequently be collapsed. We first present the overall skeleton (Algorithm 4) for treating

ECs "on-the-fly", which consists of two parts: (i) identification of ECs; and (ii) processing them. The instantiations for BRTDP and DQL will differ in the identification phase. Hence, before proceeding to the individual identification algorithms, we first establish the correctness of the processing phase.

Algorithm 4 Extension for general MDPs

```
 1: function ON-THE-FLY-EC
 2:     $\mathcal{M} \leftarrow$ IDENTIFYECS                          ▷ IDENTIFICATION OF ECS
 3:     for all $(R, B) \in \mathcal{M}$ do                            ▷ PROCESS ECS
 4:         COLLAPSE$(R, B)$
 5:         for all $s \in R$ and $a \in E(s) \setminus B$ do
 6:             $U(s_{(R,B)}, a) \leftarrow U(s, a)$
 7:             $L(s_{(R,B)}, a) \leftarrow L(s, a)$
 8:         if $R \cap F \neq \emptyset$ then
 9:             MAKETERMINAL$(s_{(R,B)}, 1)$
10:         else if no actions enabled in $s_{(R,B)}$ then
11:             MAKETERMINAL$(s_{(R,B)}, 0)$
```

Lemma 2. *Assume (R, B) is an EC in MDP M, V_{M} the value before the* PROCESS ECS *procedure in Algorithm 4, and $V_{\mathsf{M}'}$ the value after the procedure, then:*
- *for $i \in \{0, 1\}$ if* MAKETERMINAL$(s_{(R,B)}, i)$ *is called, then $\forall s \in R : \ V_{\mathsf{M}}(s) = i$,*
- *$\forall s \in S \setminus R : \ V_{\mathsf{M}}(s) = V_{\mathsf{M}'}(s)$,*
- *$\forall s \in R : \ V_{\mathsf{M}}(s) = V_{\mathsf{M}'}(s_{(R,B)})$.*

Interpretation of collapsing. Intuitively, once an EC (R, B) is collapsed, the algorithm in the EXPLORE phase can choose a state $s \in R$ and action $a \in E(s) \setminus B$ to leave the EC. This is simulated in the EXPLORE phase by considering all actions of the EC uniformly at random until s is reached, and then action a is chosen. Since (R, B) is an EC, playing all actions of B uniformly at random ensures s is almost surely reached. Note that the steps made inside a collapsed EC do not count towards the length of the explored path.

Now, we present the on-the-fly versions of BRTDP and DQL. For each case, we describe: (i) modification of Algorithm 1; (ii) identification of ECs; and (iii) correctness.

4.1 Complete information (BRTDP)

Modification of Algorithm 1. To obtain BRTDP working with unrestricted MDPs, we modify Algorithm 1 as follows: for iteration i of the EXPLORE phase, we insert a check after line 9 such that, if the length of the path ω explored (i.e., the number of states) is k_i (see below), then we invoke the ON-THE-FLY-EC function for BRTDP. The ON-THE-FLY-EC function possibly modifies the MDP by processing (collapsing) some ECs as described in Algorithm 4. After the ON-THE-FLY-EC function terminates, we interrupt the current EXPLORE phase, and start the EXPLORE phase for the $i{+}1$-th iteration (i.e., generating a new path again, starting from $\overline{s}$ in the modified MDP). To complete the description we describe the choice of k_i and identification of ECs.

Choice of k_i. Because computing ECs can be expensive, we do not call ON-THE-FLY-EC every time a new state is explored, but only after every k_i steps of the repeat-until loop at lines 6–10 in iteration i. The specific value of k_i can be decided experimentally and change as the computation progresses. A theoretical bound for k_i to ensure that there is an EC with high probability can be obtained from Lemma 1.

Identification of ECs. Given the current explored path ω, let (T, G) be $Appear(\omega, 0)$, that is, the set of states and actions explored in ω. To obtain the ECs from the set T of explored states, we use Algorithm 5. This computes an auxiliary MDP $\mathsf{M}^T = \langle T', \overline{s}, A', E', \Delta' \rangle$ defined as follows:

- $T' = T \cup \{t \mid \exists s \in T, a \in E(s)$ such that $\Delta(s,a)(t) > 0\}$,
- $A' = \bigcup_{s \in T} E(s) \cup \{\perp\}$,
- $E'(s) = E(s)$ if $s \in T$ and $E'(s) = \{\perp\}$ otherwise,
- $\Delta'(s,a) = \Delta(s,a)$ if $s \in T$, and $\Delta'(s,\perp)(s) = 1$ otherwise.

It then computes all MECs of M^T that are contained in T and identifies them as ECs. The following lemma states that each of these is indeed an EC in the original MDP.

Algorithm 5 Identification of ECs for BRTDP

```
1: function IdentifyECs($\mathsf{M}$, $T$)
2:     compute $\mathsf{M}^T$
3:     $\mathcal{M}' \leftarrow$ MECs of $\mathsf{M}^T$
4:     $\mathcal{M} \leftarrow \{(R,B) \in \mathcal{M}' \mid R \subseteq T\}$
```

Lemma 3. *Let M, M^T be the MDPs from the construction above and T be the set of explored states. Then every MEC (R, B) in M^T such that $R \subseteq T$ is an EC in M.*

Finally, we establish that the modified algorithm, which we refer to as *on-the-fly BRTDP*, almost surely converges; the proof is an extension of Theorem 1.

Theorem 3. *On-the-fly BRTDP converges almost surely for all MDPs.*

Example 2. Let us describe the execution of the on-the-fly BRTDP on the MDP M from Fig. 1 (left). Choose $k_i \geq 6$ for all i. The loop at lines 6 to 10 of Algorithm 1 generates a path ω that contains some (possibly zero) number of loops $m_1\, a\, m_2 b$ followed by $m_1\, a\, m_2\, c\, m_3\, d\, t$ where $t \in \{0, 1\}$. In the subsequent UPDATE phase, we set $U(m_3, d) = L(m_3, d) = 0.5$ and then $U(m_2, c) = L(m_2, c) = 0.5$; none of the other values change. In the second iteration of the loop at lines 6 to 10, the path $\omega' = m_1\, a\, m_2\, b\, m_1\, a\, m_2\, b \ldots$ is being generated, and the newly inserted check for ON-THE-FLY-EC will be triggered once ω achieves the length k_i.

The algorithm now aims to identify ECs in the MDP based on the part of the MDP explored so far. To do so, the MDP M^T for the set $T = \{m_1, m_2\}$ is constructed and we depict it in Fig. 1 (centre). We then run MEC detection on M^T, finding that $(\{m_1, m_2\}, \{a, b\})$ is an EC, and so it gets collapsed according to the COLLAPSE procedure. This gives the MDP M' from Fig. 1 (right).

The execution then continues with M'. A new path is generated at lines 6 to 10 of Algorithm 1; suppose it is $\omega'' = s_{\mathcal{C}} c m_3 d 0$. In the UPDATE phase we then update the value $U(s_{\mathcal{C}}, d) = L(s_{\mathcal{C}}, d) = 0.5$, which makes the condition at the last line of Algorithm 1 satisfied, and the algorithm finishes, having computed the correct value.

4.2 Limited information (DQL)

Modification of Algorithm 1 and identification of ECs. The modification of Algorithm 1 is done exactly as for the modification of BRDTP (i.e., we insert a check after line 9 of EXPLORE, which invokes the ON-THE-FLY-EC function if the length of path ω exceeds k_i). In iteration i, we set k_i as $3\ell_i^3$, for some ℓ_i (to be described later). The identification of the EC is as follows: we consider $\mathit{Appear}(\omega, \ell_i)$, the set of states and actions that have appeared more than ℓ_i times in the explored path ω, which is of length $3\ell_i^3$, and identify the set as an EC; i.e., $\mathcal{M}$ in line 2 of Algorithm 4 is defined as the set containing the single tuple $\mathit{Appear}(\omega, \ell_i)$. We refer to the algorithm as *on-the-fly DQL*.

Choice of ℓ_i and correctness. The choice of ℓ_i is as follows. Note that, in iteration i, the error probability, obtained from Lemma 1, is at most $2c^{\ell_i}\ell_i^3 \cdot (p_{\min}/E_m)^{-\kappa}$ and we choose ℓ_i such that $2c^{\ell_i}\ell_i^3 \cdot (p_{\min}/E_m)^{-\kappa} \le \frac{\delta/2}{2^i}$, where δ is the confidence. Note that, since $c < 1$, we have that c^{ℓ_i} decreases exponentially, and hence for every i such ℓ_i exists. It follows that the total error of the algorithm due to the on-the-fly EC collapsing is at most $\delta/2$. It follows from the proof of Theorem 2 that for on-the-fly DQL the error is at most δ if we use the same $\bar{\varepsilon}$ as for DQL, but now with DQL confidence $\delta/4$, i.e., with $m = \frac{\ln(24|S||A|(1+\frac{|S||A|}{\bar{\varepsilon}})/\delta)}{2\bar{\varepsilon}^2}$. As before, these numbers can be conservatively approximated using the limited information.

Theorem 4. *On-the-fly DQL is probably approximately correct for all MDPs.*

Example 3. Let us now briefly explain the execution of on-the-fly DQL on the MDP M from Fig. 1 (left). At first, paths of the same form as ω in Example 2 will be generated and there will be no change to U and L, because in any call to UPDATE (see Algorithm 3) for states $s \in \{m_1, m_2\}$ with $c(s,a) = m$ the values accumulated in $\mathit{accum}_m^U(s,a)/m$ and $\mathit{accum}_m^L(s,a)/m$ are the same as the values already held, namely 1 and 0, respectively.

At some point, we call UPDATE for the tuple (m_3, d) with $c(m_3, d) = m$, which will result in the change of $U(m_3, d)$ and $L(m_3, d)$. Note, that at this point, the numbers $\mathit{accum}_m^U(s,d)/m$ and $\mathit{accum}_m^L(s,d)/m$ are both equal to the proportion of generated paths that visited the state 1. This number will, with high probability, be very close to 0.5, say 0.499. We thus set $U(m_3, d) = 0.499 + \varepsilon$ and $L(m_3, d) = 0.499 - \varepsilon$.

We then keep generating paths of the same form and at some point also update $U(m_2, c)$ and $L(m_2, c)$ to precisely $0.499 + \varepsilon$ and $0.499 - \varepsilon$, respectively. The subsequently generated path will be looping on m_1 and m_2, and once it is of length ℓ_i, we identify $(\{m_1, m_2\}, \{a, b\})$ as an EC due to the definition of $\mathit{Appear}(\omega, \ell_i)$. We then get the MDP from Fig. 1 (right), which we use to generate new paths, until the upper and lower bounds on value in the new initial state are within the required bound.

4.3 Extension to LTL

So far we have focused on reachability, but our techniques also extend to linear temporal logic (LTL) objectives. By translating an LTL formula to an equivalent deterministic ω-automaton, verifying MDPs with LTL objectives reduces to analysis of MDPs with ω-regular conditions such as Rabin acceptance conditions. A Rabin acceptance condition consists of a set $\{(M_1, N_1) \dots (M_d, N_d)\}$ of d pairs (M_i, N_i), where each $M_i \subseteq S$ and

$N_i \subseteq S$. The acceptance condition requires that, for some $1 \leq i \leq d$, states in M_i are visited infinitely often and states in N_i are visited finitely often.

Value computation for MDPs with Rabin objectives reduces to optimal reachability of *winning* ECs, where an EC (R, B) is winning if $R \cap M_i \neq \emptyset$ and $R \cap N_i = \emptyset$ for some $1 \leq i \leq d$ [12]. Thus, extending our results from reachability to Rabin objectives requires processing of ECs for Rabin objectives (line 3-11 of Algorithm 4), which is done as follows. Once an EC (R, B) is identified, we first obtain the EC in the original MDP (i.e., obtain the set of states and actions corresponding to the EC in the original MDP) as $(\overline{R}, \overline{B})$ and then determine if there is a sub-EC of $(\overline{R}, \overline{B})$ that is winning using standard algorithms for MDPs with Rabin objectives [2]; and if so then we merge the whole EC as in line 9 of Algorithm 4; if not, and, moreover, there is no action out of the EC, we merge as in line 11 of Algorithm 4. This modified EC processing yields on-the-fly BRTDP and DQL algorithms for MDPs with Rabin objectives.

5 Experimental Results

Implementation. We have developed an implementation of our learning-based framework within the PRISM model checker [25], building upon its simulation engine for generating trajectories and explicit probabilistic model checking engine for storing visited states and U and L values. We focus on the complete-information case (i.e., BRTDP), for which we can perform a more meaningful comparison with PRISM. We implement Algorithms 1 and 2, and the on-the-fly EC detection algorithm of Sec. 4, with the optimisation of taking T as the set of all states explored so far.

We consider three distinct variants of the learning algorithm by modifying the GETSUCC function in Algorithm 1, which is the heuristic responsible for picking a successor state s' after choosing some action a in each state s of a trajectory. The first variant takes the unmodified GETSUCC, selecting s' at random according to the distribution $\Delta(s, a)$. This behaviour follows the one of the original RTDP algorithm [3]. The second uses the heuristic proposed for BRTDP in [30], selecting the successor $s' \in supp(\Delta(s, a))$ that maximises the difference $U(s') - L(s')$ between bounds for those states (M-D). For the third, we propose an alternative approach that systematically chooses all successors s' in a round-robin (R-R) fashion, and guarantees termination with certainty.

Results. We evaluated our implementation on four existing benchmark models, using a machine with a 2.8GHz Xeon processor and 32GB of RAM, running Fedora 14. We use three models from the PRISM benchmark suite [26]: *zeroconf*, *wlan*, and *firewire_impl_dl*; and a fourth one from [16]: *mer*. The first three use unbounded probabilistic reachability properties; the fourth a time-bounded probabilistic reachability. The latter is used to show differences between heuristics in the case of MDPs containing rare events, e.g., MDPs where failures occur with very low probability. All models, properties and logs are available online at [39].

We run BRTDP and compare its performance to PRISM. We terminate it when the bounds L and U differ by at most ε for the initial state of the MDP. We use $\varepsilon = 10^{-6}$ in all cases except *zeroconf*, where $\varepsilon = 10^{-8}$ is used since the actual values are very small. For PRISM, we use its fastest engine, which is the "sparse" engine, running value iteration. This is terminated when the values for all states in successive iterations differ

Name [param.s]	Param. values	Num. states	Time (s) PRISM	RTDP	M-D	R-R	Visited states RTDP	M-D	R-R
zeroconf $[N, K]$	$20, 10$	3,001,911	129.9	7.40	1.47	1.83	760	2007	2570
	$20, 14$	4,427,159	218.2	12.4	2.18	2.26	977	3728	3028
	$20, 18$	5,477,150	303.8	71.5	3.89	3.73	1411	5487	3704
wlan $[BOFF]$	4	345,000	7.35	0.53	0.48	0.54	2018	1377	1443
	5	1,295,218	22.3	0.55	0.45	0.54	2053	1349	1542
	6	5,007,548	82.9	0.50	0.43	0.49	1995	1313	1398
firewire_impl_dl $[delay, deadline]$	$36, 200$	6,719,773	63.8	2.85	2.62	2.26	26,508	28,474	22,038
	$36, 240$	13,366,666	145.4	8.37	7.69	6.72	25,214	26,680	20,219
	$36, 280$	19,213,802	245.4	9.29	7.90	7.39	32,214	28,463	25,565
mer $[N, q]$	$3000, 0.0001$	17,722,564	158.5	67.0	2.42	4.44	1950	3116	3729
	$3000, 0.9999$	17,722,564	157.7	10.9	2.82	6.80	2902	4643	4608
	$4500, 0.0001$	26,583,064	250.7	67.3	2.41	4.42	1950	3118	3729
	$4500, 0.9999$	26,583,064	246.6	10.9	2.84	6.79	2900	4644	4608

Table 1. Verification times using BRTDP (three different heuristics) and PRISM.

by at most ε. Strictly speaking, this is not guaranteed to produce an ε-optimal strategy (e.g. in the case of very slow numerical convergence), but on all these examples it does.

The experimental results are summarised in Table 1. For each model, we give the number of states in the full model, the time for PRISM (model construction, precomputation of zero/one states and value iteration) and time and number of visited states for BRTDP with each of the three heuristics described earlier. Some heuristics perform random exploration and therefore all results have been averaged over 20 runs.

We see that our method outperforms PRISM on all four benchmarks. The improvements in execution time on these benchmarks are possible because the algorithm is able to construct an ε-optimal policy whilst exploring only a portion of the state space. The number of states visited by the algorithm is at least two orders of magnitude smaller than the total size of the model (column 'Num. states'). These numbers do not vary greatly between heuristics.

The RTDP heuristic is generally the slowest of the three, and tends to be sensitive to the probabilities in the model. In the *mer* example, changing the parameter q can mean that some states, which are crucial for the convergence of the algorithm, are no longer visited due to low probabilities on incoming transitions. This results in a considerable slow-down, and is a potential problem for MDPs containing rare events. The M-D and R-R heuristics perform very similarly, despite being quite different (one is randomised, the other deterministic). Both perform consistently well on these examples.

6 Conclusions

We have presented a framework for verifying MDPs using learning algorithms. Building upon methods from the literature, we provide novel techniques to analyse unbounded probabilistic reachability properties of arbitrary MDPs, yielding either exact bounds, in the case of complete information, or PAC bounds, in the case of limited information. Given our general framework, one possible direction would be to explore other learning algorithms in the context of verification. Another direction of future work is to explore whether learning algorithms can be combined with symbolic methods for probabilistic verification.

Acknowledgement. We thank Arnd Hartmanns and anonymous reviewers for careful reading and valuable feedback.

References

1. Aljazzar, H., Leue, S.: Generation of counterexamples for model checking of Markov decision processes. In: QEST. pp. 197–206 (2009)
2. Baier, C., Katoen, J.P.: Principles of model checking. MIT Press (2008)
3. Barto, A.G., Bradtke, S.J., Singh, S.P.: Learning to act using real-time dynamic programming. Artificial Intelligence 72(12), 81 – 138 (1995)
4. Bogdoll, J., Fioriti, L.M.F., Hartmanns, A., Hermanns, H.: Partial order methods for statistical model checking and simulation. In: FMOODS/FORTE. pp. 59–74 (2011)
5. Bogdoll, J., Hartmanns, A., Hermanns, H.: Simulation and statistical model checking for modestly nondeterministic models. In: MMB/DFT. pp. 249–252 (2012)
6. Boyer, B., Corre, K., Legay, A., Sedwards, S.: PLASMA-lab: A flexible, distributable statistical model checking library. In: QEST. pp. 160–164 (2013)
7. Brázdil, T., Chatterjee, K., Chmelıik, M., Forejt, V., Křetínský, J., Kwiatkowska, M.Z., Parker, D., Ujma, M.: Verification of Markov decision processes using learning algorithms. CoRR abs/1402.2967 (2014)
8. Bulychev, P.E., David, A., Larsen, K.G., Mikucionis, M., Poulsen, D.B., Legay, A., Wang, Z.: UPPAAL-SMC: Statistical model checking for priced timed automata. In: QAPL (2012)
9. Chatterjee, K., Henzinger, M.: An O(n^2) algorithm for alternating Büchi games. In: SODA. pp. 1386–1399 (2012)
10. Chatterjee, K., Henzinger, M.: Faster and dynamic algorithms for maximal end-component decomposition and related graph problems in probabilistic verification. In: SODA (2011)
11. Ciesinski, F., Baier, C., Grosser, M., Klein, J.: Reduction techniques for model checking Markov decision processes. In: QEST. pp. 45–54 (2008)
12. Courcoubetis, C., Yannakakis, M.: Markov decision processes and regular events (extended abstract). In: ICALP. pp. 336–349 (1990)
13. David, A., Larsen, K.G., Legay, A., Mikucionis, M., Poulsen, D.B., van Vliet, J., Wang, Z.: Statistical model checking for networks of priced timed automata. In: FORMATS (2011)
14. David, A., Larsen, K.G., Legay, A., Mikucionis, M., Wang, Z.: Time for statistical model checking of real-time systems. In: CAV. pp. 349–355 (2011)
15. De Alfaro, L.: Formal verification of probabilistic systems. Ph.D. thesis (1997)
16. Feng, L., Kwiatkowska, M., Parker, D.: Automated learning of probabilistic assumptions for compositional reasoning. In: FASE. pp. 2–17 (2011)
17. He, R., Jennings, P., Basu, S., Ghosh, A.P., Wu, H.: A bounded statistical approach for model checking of unbounded until properties. In: ASE. pp. 225–234 (2010)
18. Henriques, D., Martins, J., Zuliani, P., Platzer, A., Clarke, E.M.: Statistical model checking for Markov decision processes. In: QEST. pp. 84–93 (2012)
19. Hérault, T., Lassaigne, R., Magniette, F., Peyronnet, S.: Approximate probabilistic model checking. In: VMCAI. pp. 307–329 (2004)
20. Jégourel, C., Legay, A., Sedwards, S.: Cross-entropy optimisation of importance sampling parameters for statistical model checking. In: CAV. pp. 327–342 (2012)
21. Jégourel, C., Legay, A., Sedwards, S.: A platform for high performance statistical model checking - PLASMA. In: TACAS. pp. 498–503 (2012)
22. Jégourel, C., Legay, A., Sedwards, S.: Importance splitting for statistical model checking rare properties. In: CAV. pp. 576–591 (2013)
23. Kemeny, J., Snell, J., Knapp, A.: Denumerable Markov Chains. Springer-Verlag (1976)
24. Kolobov, A., Mausam, Weld, D.S., Geffner, H.: Heuristic search for generalized stochastic shortest path mdps. In: ICAPS (2011)
25. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: CAV. pp. 585–591 (2011)

26. Kwiatkowska, M., Norman, G., Parker, D.: The PRISM benchmark suite. In: QEST. pp. 203–204 (2012)
27. Larsen, K.G.: Priced timed automata and statistical model checking. In: IFM (2013)
28. Lassaigne, R., Peyronnet, S.: Approximate planning and verification for large Markov decision processes. In: SAC. pp. 1314–1319 (2012)
29. Legay, A., Sedwards, S.: Lightweight Monte Carlo algorithm for Markov decision processes. CoRR abs/1310.3609 (2013)
30. McMahan, H.B., Likhachev, M., Gordon, G.J.: Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. In: ICML (2005)
31. Puterman, M.: Markov Decision Processes. Wiley (1994)
32. Rabih, D.E., Pekergin, N.: Statistical model checking using perfect simulation. In: ATVA. pp. 120–134 (2009)
33. Sen, K., Viswanathan, M., Agha, G.: On statistical model checking of stochastic systems. In: CAV. pp. 266–280 (2005)
34. Sen, K., Viswanathan, M., Agha, G.: Statistical model checking of black-box probabilistic systems. In: CAV. pp. 202–215 (2004)
35. Strehl, A.L., Li, L., Wiewiora, E., Langford, J., Littman, M.L.: PAC model-free reinforcement learning. In: ICML. pp. 881–888 (2006)
36. Sutton, R., Barto, A.: Reinforcement Learning: An Introduction. MIT Press (1998)
37. Younes, H., Simmons, R.: Probabilistic verification of discrete event systems using acceptance sampling. In: CAV. pp. 223–235 (2002)
38. Younes, H.L.S., Clarke, E.M., Zuliani, P.: Statistical verification of probabilistic properties with unbounded until. In: SBMF. pp. 144–160 (2010)
39. `http://www.prismmodelchecker.org/files/atva14learn/`

Counterexample Explanation by Learning Small Strategies in Markov Decision Processes

Tomáš Brázdil[1], Krishnendu Chatterjee[2], Martin Chmelík[2], Andreas Fellner[2], and Jan Křetínský[2]

[1] Masaryk University, Brno, Czech Republic
[2] IST Austria

Abstract. For deterministic systems, a counterexample to a property can simply be an error trace, whereas counterexamples in probabilistic systems are necessarily more complex. For instance, a set of erroneous traces with a sufficient cumulative probability mass can be used. Since these are too large objects to understand and manipulate, compact representations such as subchains have been considered. In the case of probabilistic systems with non-determinism, the situation is even more complex. While a subchain for a given strategy (or scheduler, resolving non-determinism) is a straightforward choice, we take a different approach. Instead, we focus on the strategy itself, and extract the most important decisions it makes, and present its succinct representation.

The key tools we employ to achieve this are (1) introducing a concept of importance of a state w.r.t. the strategy, and (2) learning using decision trees. There are three main consequent advantages of our approach. Firstly, it exploits the quantitative information on states, stressing the more important decisions. Secondly, it leads to a greater variability and degree of freedom in representing the strategies. Thirdly, the representation uses a self-explanatory data structure. In summary, our approach produces more succinct and more explainable strategies, as opposed to e.g. binary decision diagrams. Finally, our experimental results show that we can extract several rules describing the strategy even for very large systems that do not fit in memory, and based on the rules explain the erroneous behaviour.

1 Introduction

The standard models for dynamic stochastic systems with both probabilistic and non-deterministic behaviour are *Markov decision processes* (MDPs) [How60,Put94,FV97]. They are widely used in verification of probabilistic systems [BK08,KNP11] in several ways. Firstly, in concurrent probabilistic systems, such as communication protocols, the nondeterminism arises from scheduling [CY95,Var85]. Secondly, in probabilistic systems operating in open environments, such as various stochastic reactive systems, nondeterminism arises from environmental inputs [Seg95,dA97]. Thirdly, for underspecified probabilistic systems, a controller is synthesized, resolving the nondeterminism in a way that optimizes some objective, such as energy consumption or time constraints in embedded systems [BK08,KNP11].

In analysis of MDPs, the behaviour under all possible strategies (schedulers, controllers, policies) is examined. For example, in the first two cases, the result of the verification process is either a guarantee that a given property holds under all strategies, or a counterexample strategy. In the third case, either a witness strategy guaranteeing a given property is synthesized, or its non-existence is stated. In all settings, it is desirable that the output *strategies should be "small and understandable"* apart from correct. Intuitively, it is a strategy with a representation small enough for the human debugger to read and understand where the bug is (in the verification setting), or for the programmer to implement in the device (in the synthesis setting). In this paper, we focus on the verification setting and illustrate our approach mainly on probabilistic protocols. Nonetheless, our results immediately carry over to the synthesis setting.

Obtaining a small and simple strategy may be impossible if the strategy is required to be optimal, i.e., in our setting reaching the error state with the highest possible probability. Therefore, there is a trade-off between simplicity and optimality of the strategies. However, in order to debug a system, a simple counterexample or a series thereof is more valuable than the most comprehensive, but incomprehensible counterexample. In practice, a simple strategy reaching the error with probability smaller by a factor of ε, e.g. one per cent, is a more valuable source of information than a huge description of an optimal strategy. Similarly, controllers in embedded devices should strive for optimality, but only as long as they are small enough to fit in the device. In summary, we are interested in finding small and simple close-to-optimal (ε-optimal) strategies.

How can one obtain a small and simple strategy? This seems to require some understanding of the particular system and the bug. How can we do something like that automatically? The approaches have so far been limited to BDD representations of the strategy, or generating subchains representing a subset of paths induced by the strategy. While BDDs provide a succinct representation, they are not well readable and understandable. Further, subchains do not focus on the decisions the strategy makes at all. In contrast, a huge effort has been spent on methods to obtain "understanding" from large sets of data, using *machine learning* methods. In this paper, we propose to extend their use in verification, namely of reachability properties in MDPs, in several ways. Our first aim of using these methods is to efficiently exploit the structure that is present in the models, written in e.g. PRISM language with variables and commands. This structure gets lost in the traditional numerical analysis of the MDPs generated from the PRISM language description. The second aim is to distil more information from the generated MDPs, namely the importance of each decision. Both lead to an improved understanding of the strategy's decisions.

Our approach. We propose three steps to obtain the desired strategies. Each of them has a positive effect on the resulting size.

(1) Obtaining a (possibly partially defined and liberal) ε-optimal strategy. The ε-optimal strategies produced by standard methods, such as value iteration of PRISM [KP13], may be too large to compute and overly specific. Firstly, as argued in [BCC$^+$14], typically only a small fraction of the system needs to be explored in order to find an ε-optimal strategy, whereas most states are reached with only a very small probability. Without much loss, the strategy may not be defined there. For example, in the MDP M depicted in Fig. 1, the decision in q (and v_i's) is almost irrelevant for the overall probability of reaching t from s. Such a partially defined strategy can be obtained using learning methods [BCC$^+$14].

Secondly, while the usual strategies prescribe which action to play, *liberal* strategies leave more choices open. There are several advantages of liberal strategies, and similar notions of strategies called permissive strategies have been studied

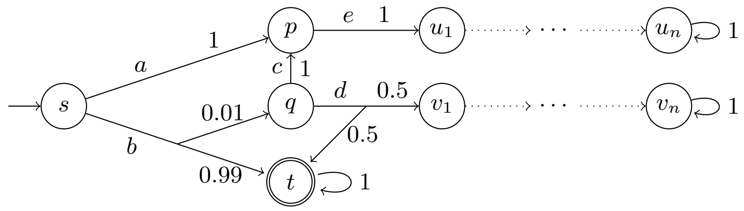


Fig. 1: An MDP M with reachability objective t

in [BJW02,BMOU11,DFK$^+$14]. A liberal strategy, instead of choosing an action in each state, chooses a set of actions to be played uniformly at every state. First, each liberal strategy represents a set of strategies, and thus covers more behaviour. Second, in counter-example guided abstraction-refinement (CEGAR) analysis, since liberal strategies can represent sets of counter-examples, they accelerate the abstraction-refinement loop by ruling out several counter-examples at once. Finally, they also allow for more robust learning of smaller strategies in Step 3. We show that such strategies can be obtained from standard value iteration as well as [BCC$^+$14]. Further processing of the strategies in Step 2 and 3 allows liberal strategies as input and preserves liberality in the small representation of the strategy.

(2) Identifying important parts of the strategy. We define a concept of *importance* of a state w.r.t. a strategy, corresponding to the probability of visiting the state by the strategy. Observe that only a fraction of states can be reached while following the strategy, and thus have positive importance. On the unreachable states, with zero importance, the definition of the strategy is useless. For instance, in M, both states p and q must have been explored when constructing the strategy in order to find out whether it is better to take action a or b. However, if the resulting strategy is to use b and d, the information what to do in u_i's is useless. In addition, we consider v_i's to be of zero importance, too, since they are never reached on the way to target.

Furthermore, apart from ignoring states with zero importance, we want to partially ignore decisions that are unlikely to be made (in less important states such as q), and in contrast, stress more the decisions in important states likely to be visited (such as s). Note that this is difficult to achieve in data structures that remember all the stored data exactly, such as BDDs. Of course, we can store decisions in states with importance above a certain threshold. However, we obtain much smaller representations if we allow more variability and reflect the whole quantitative information, as shown in Step 3.

(3) Data structures for compact representation of strategies. The explicit representation of a strategy by a table of pairs (state, action to play) results in a huge amount of data since the systems often have millions of states. Therefore, a symbolic representation by binary decision diagrams (BDD) looks as a reasonable option. However, there are several drawbacks of using BDDs. Firstly, due to the bit-level representation of the state-action pairs, the resulting BDD is not very readable. Secondly, it is often still too large to be understood by human, for instance due to a bad ordering of the variables. Thirdly, it cannot quantitatively reflect the differences in the importance of states.

Therefore, we propose to use *decision trees* instead , e.g. [Mit97], a structure similar to BDDs, but with nodes labelled by various predicates over the system's variables. They have several advantages. Firstly, they yield an explanation of the decision, as opposed to e.g. neural networks, and thus provide an explanation how the strategy works. Secondly, sophisticated algorithms for their construction, based on entropy, result in smaller representation than BDD, where a good ordering of variables is known to be notoriously difficult to find [BK08]. Thirdly, as suggested in Step 2, they allow for less probable remembering of less stressed data if this sufficiently simplifies the tree and decreases its size. Finally, the major drawback of decision trees in machine learning—frequent overfitting of the training data—is not an issue in our setting since the tree is not used for classification of test data, but only of the training data.

Summary of our contribution. In summary our contributions are as follows:

- We provide a method for obtaining succinct representation of ε-optimal strategies as decision trees. The method is based on a new concept of importance measure and on well-established machine learning techniques.
- Experimental data show that even for some systems larger than the available memory, our method yields trees with only several dozens of nodes.
- We illustrate the understandability of the representation on several examples from PRISM benchmarks [KNP12], reading off respective bug explanations.

Related work. In artificial intelligence, compact (factored) representations of MDP structure have been developed using dynamic Bayesian networks [BDG95,KK99], probabilistic STRIPS [KHW94], algebraic decision diagrams [HSaHB99], and also decision trees [BDG95]. Formalisms used to represent MDPs can, in principle, be used to represent values and policies as well. In particular, variants of decision trees are probably the most used [BDG95,CK91,KP99]. For a detailed survey of compact representations see [BDH99]. In the context of verification, MDPs are often represented using variants of (MT)BDDs [dAKN$^+$00,HKN$^+$03,MP04], and strategies by BDDs [WBB$^+$10].

Decision trees have been used in connection with real-time dynamic programming and reinforcement learning [BD96,Pye03]. Learning a compact decision tree representation of a policy has been investigated in [SLT10] for the case of body sensor networks, but the paper aims at a completely different application field (a simple model of sensor networks as opposed to generic PRISM models), uses different objectives (discounted rewards), and does not consider the importance of a state that, as we show, may substantially decrease sizes of resulting policies.

Our results are related to the problem of computing minimal/small counterexamples in probabilistic verification. Most papers concentrate on solving this problem for Markov chains and linear-time properties [HKD09,ADvR08,WJÁ$^+$14,JÁK$^+$11], branching-time properties [DHK08,FHPW10,AL10], and in the context of simulation [KPC12]. A couple of tools have been developed for probabilistic counterexample generation, namely DiPro [ALFLS11] and COMICS [JÁV$^+$12]. For a detailed survey see [ÁBD$^+$14]. While previous approaches focus on presenting diagnostic paths forming the counterexample, our approach focuses on decisions made by the respective strategy.

Concerning MDPs, [WJÁ$^+$14] uses mixed integer linear programming to compute minimal critical sub-systems, i.e. whole sub-MDPs as opposed to a compact representation of "right" decisions computed by our methods. [AL09] uses a directed on-the-fly search to compute sets of most probable diagnostic paths (which somehow resembles our notion of importance), but the paths are encoded explicitly by AND/OR trees as opposed to our use of decision trees. Neither of these papers takes advantage of an internal structure of states and their methods substantially differ from ours. The notion of paths encoded as AND/OR trees has also been studied in [LL13] to represent probabilistic counter-examples visually as fault trees, and then derive causal (the cause and effect) relationship between events. [KH09] develops abstraction-based framework for model-checking MDPs based on games, which allows to trade compactness for precision, but does not give a procedure for constructing (a compact representation of) counterexample strategies. [WJV$^+$13,DJW$^+$14] computes a smallest set of guarded commands (of a PRISM-like language) that induce a critical subsystem, but, unlike our methods, does not provide a compact representation of actual decisions needed to reach an erroneous state; moreover, there is not always a command based counterexample.

Counter-examples play a crucial role in CEGAR analysis of MDPs, and have been widely studied, such as, game-based abstraction refinement [KNP06]; non-compositional CEGAR approach for reachability [HWZ08] and safe-pCTL [CV10]; compositional CEGAR approach for safe-pCTL and qualitative logics [KPC12,CCD14]; and abstraction-refinement for quantitative properties [DJJL01,DJJL02]. All of these works only consider a single strategy represented explicitly, whereas our approach considers a succinct representation of a set of strategies, and can accelerate the abstraction-refinement loop.

2 Preliminaries

We use $\mathbb{N}$, $\mathbb{Q}$, and $\mathbb{R}$ to denote the sets of positive integers, rational and real numbers, respectively. The set of all rational probability distributions over a finite set X is denoted by $\mathit{Dist}(X)$. Further, $d \in \mathit{Dist}(X)$ is Dirac if $d(x) = 1$ for some $x \in X$. Given a function $f : X \to \mathbb{R}$, we write $\arg\max_{x \in X} f(x) = \{x \in X \mid f(x) = \max_{x' \in X} f(x')\}$.

Markov chains. A *Markov chain* is a tuple $M = (L, P, \mu)$ where L is a finite set of locations, $P : L \to \mathit{Dist}(L)$ is a probabilistic transition function, and $\mu \in \mathit{Dist}(L)$ is the initial probability distribution. We denote the respective unique probability measure for M by $\mathbb{P}$.

Markov decision processes. A *Markov decision process* (MDP) is a tuple $G = (S, A, \mathit{Act}, \delta, \hat{s})$ where S is a finite set of states, A is a finite set of actions, $\mathit{Act} : S \to 2^A \setminus \{\emptyset\}$ assigns to each state s the set $\mathit{Act}(s)$ of actions enabled in s, $\delta : S \times A \to \mathit{Dist}(S)$ is a probabilistic transition function that, given a state and an action, gives a probability distribution over the successor states, and $\hat{s}$ is the initial state. A *run* in G is an infinite alternating sequence of states and actions $\omega = s_1 a_1 s_2 a_2 \cdots$ such that for all $i \geq 1$, we have $a_i \in \mathit{Act}(s_i)$ and $\delta(s_i, a_i)(s_{i+1}) > 0$. A *path* of length k in G is a finite prefix $w = s_1 a_1 \cdots a_{k-1} s_k$ of a run in G.

Strategies and plays. Intuitively, a strategy (or a policy) in an MDP G is a “recipe” to choose actions. Formally, a strategy is a function $\sigma : S \to \mathit{Dist}(A)$ that given the current state of a play gives a probability distribution over the enabled actions.[1] In general, a strategy may randomize, i.e. return non-Dirac distributions. A strategy is *deterministic* if it gives a Dirac distribution for every argument.

[1] In general, a strategy may be history dependent. However, for objectives considered in this paper, *memoryless* strategies (depending on the last state visited) are sufficient. Therefore, we only consider memoryless strategies in this paper.

A *play* of G determined by a strategy σ and a state $\bar{s} \in S$ is a Markov chain $G_{\bar{s}}^{\sigma}$ where the set of locations is S, the initial distribution μ is Dirac with $\mu(\bar{s}) = 1$ and

$$P(s)(s') = \sum_{a \in A} \sigma(s)(a) \cdot \delta(s,a)(s') \,.$$

The induced probability measure is denoted by $\mathbb{P}_{\bar{s}}^{\sigma}$ and "almost surely" or "almost all runs" refers to happening with probability 1 according to this measure. We usually write $\mathbb{P}^{\sigma}$ instead of $\mathbb{P}_{\hat{s}}^{\sigma}$ (here $\hat{s}$ is the initial state of G).

Liberal strategies. A *liberal strategy* is a function $\varsigma : S \to 2^A$ such that for every $s \in S$ we have that $\emptyset \neq \varsigma(s) \subseteq \mathit{Act}(s)$. Given a liberal strategy ς and a state s, an action $a \in \mathit{Act}(s)$ is *good* (in s w.r.t. ς) if $a \in \varsigma(s)$, and *bad* otherwise. Abusing notation, we denote by ς the strategy that to every state s assigns the uniform distribution on $\varsigma(s)$ (which, in particular, allows us to use G_s^{ς}, $\mathbb{P}_s^{\varsigma}$ and apply the notion of ε-optimality to ς).

Reachability objectives. Given a set $F \subseteq S$ of *target states*, we denote by $\Diamond F$ the set of all runs that visit a state of F. For a state $s \in S$, the *maximal reachability probability* (or simply *value*) in s, is $\mathit{Val}(s) := \max_{\sigma} \mathbb{P}_s^{\sigma}[\Diamond F]$. Given $\epsilon \geq 0$, we say that a strategy σ is *ε-optimal* if $\mathbb{P}^{\sigma}[\Diamond F] \geq \mathit{Val}(\hat{s}) - \varepsilon$, and we call a 0-optimal strategy *optimal*.[2] To avoid overly technical notation, we assume that states of F, subject to the reachability objective, are absorbing, i.e. for all $s \in F, a \in \mathit{Act}(s)$ we have $\delta(s,a)(s) = 1$.

End components. A non-empty set $S' \subseteq S$ is an *end component* (EC) of G if there is $\mathit{Act}' : S' \to 2^A \setminus \{\emptyset\}$ such that (1) for all $s \in S'$ we have $\mathit{Act}'(s) \subseteq \mathit{Act}(s)$, (2) for all $s \in S'$, we have $a \in \mathit{Act}'(s)$ iff $\delta(s,a) \in \mathit{Dist}(S')$, and (3) for all $s, t \in S'$ there is a path $\omega = s_1 a_1 \cdots a_{k-1} s_k$ such that $s_1 = s$, $s_k = t$, and $s_i \in S', a_i \in \mathit{Act}'(s_i)$ for every i. An end component is a *maximal end component* (MEC) if it is maximal with respect to the subset ordering. Given an MDP, the set of MECs is denoted by MEC. Given a MEC, actions of $\mathit{Act}'(s)$ and $\mathit{Act}(s) \setminus \mathit{Act}'(s)$ are called *internal* and *external* (in state s), respectively.

3 Computing ε-optimal Strategies

There are many algorithms for solving quantitative reachability in MDPs, such as the value iteration, the strategy improvement, linear programming based methods etc., see [Put94]. The main method implemented in PRISM is the value iteration, which successively (under)approximates the value $\mathit{Val}(s,a) = \sum_{s' \in A} \delta(s,a)(s') \cdot \mathit{Val}(s')$ of every state-action pair (s,a) by a value $V(s,a)$, and stops when the approximation is good enough. Denoting by $V(s) := \max_{a \in \mathit{Act}(s)} V(s,a)$, every step of the value iteration *improves* the approximation $V(s,a)$ by assigning $V(s,a) := \sum_{s' \in S} \delta(s,a)(s') \cdot V(s')$ (we start with V such that $V(s) = 1$ if $s \in F$, and $V(s) = 0$ otherwise).

The disadvantage of the standard value iteration (and also most of the above mentioned traditional methods) is that it works with the whole state space of the MDP (or at least with its reachable part). For instance, consider states u_i, v_i of Fig. 1. The paper [BCC$^+$14] adapts methods of bounded real-time dynamic programming (BRTDP, see e.g. [MLG05]) to speed up the computation of the value iteration by improving $V(s,a)$ [3] only on "important" state-action pairs identified by simulations.

[2] For every MDP, there is a memoryless deterministic optimal strategy, see e.g. [Put94].

[3] Here we use V for the lower approximation denoted by V_L in [BCC$^+$14].

Even though RTDP methods may substantially reduce the size of an ε-optimal strategy, its explicit representation is usually large and difficult to understand. Thus we develop succinct representations of strategies, based on decision trees, that will reduce the size even further and also provide a human readable representation. Even though the above methods are capable of yielding *deterministic* ε-optimal strategies, that can be immediately fed into machine learning algorithms, we found it advantageous to give the learning algorithm more freedom in the sense that if there are more ε-optimal strategies, we let the algorithm choose (uniformly). This is especially useful within MECs where many actions have the same value. Therefore, we extract *liberal* ε-optimal strategies from the value approximation V, output either by the value iteration or BRTDP.

Computing liberal ε-optimal strategies. Let us show how to obtain a liberal strategy ς from the value iteration, or BRTDP. For simplicity, we start with MDP without MECs.

MDP without end components. We say that $V : S \times A \to [0,1]$ is a *valid ε-underapproximation* if the following conditions hold:

1. $V(s,a) \leq \mathit{Val}(s,a)$ for all $s \in S$ and $a \in A$
2. $\mathit{Val}(\hat{s}) - V(\hat{s}) \leq \varepsilon$
3. $V(s,a) \leq \sum_{s' \in S} \delta(s,a)(s') \cdot V(s')$ for all $s \in S$ and $a \in Acts$

The outputs V of both the value iteration, and BRTDP are valid ε-underapproximations. We define a liberal strategy ς^V by $\varsigma^V(s) = \arg\max_{a \in Act(s)} V(s,a)$ for all $s \in S$.[4]

Lemma 1. *For every $\varepsilon > 0$ and a valid ε-underapproximation V, ς^V is ε-optimal.* [5]

General MDP. For MDPs with end components we have to extend the definition of the valid ε-underapproximation. Given a MEC $S' \subseteq S$, we say that $(s,a) \in S \times A$ is *maximal-external in* S' if $s \in S'$, $a \in Act(s)$ is external and $V(s,a) \geq V(s',a')$ for all $s' \in S'$ and $a' \in Act(s')$. A state $s' \in S'$ is an *exit* (of S') if (s,a) is maximal-external in S' for some $a \in Act(s)$. We add the following condition to the valid ε-underapproximation:

4. Each MEC $S' \subseteq S$ has at least one exit.

Now the definition of ς^V is also more complicated:

- For every $s \in S$ which is *not* in any MEC, we put $\varsigma^V(s) = \arg\max_{a \in Act(s)} V(s,a)$.
- For every $s \in S$ which *is* in a MEC S',
  - if s is an exit, then $\varsigma^V(s) = \{a \in Act(s) \mid (s,a)$ is maximal-external in $S'\}$
  - otherwise, $\varsigma^V(s) = \{a \in Act(s) \mid a$ is internal$\}$

Using these extended definitions, Lemma 1 remains valid. Further, note that $\varsigma^V(s)$ is defined even for states with trivial underapproximation $V(s) = 0$, for instance a state s that was never subject to any value iteration improvement. Then the values $\varsigma(s)$ may not be stored explicitly, but follow implicitly from *not* storing any $V(s)$, thus assuming $V(s,\cdot) = 0$.

[4] Furthermore, one could consider liberal strategies playing also ε-optimal actions. However, our experiments did not prove better performance.

[5] Intuitively this means that randomizing among good actions of ε-optimal strategies preserves ε-optimality in the reachability setting (in contrast to other settings, e.g. with parity objectives).

4 Importance of Decisions

Note that once we have computed an ε-optimal liberal strategy ς, we may, in principle, compute a compact representation of ς (using e.g. BDDs), and obtain a strategy with possibly smaller representation than above.

However, we go one step further as follows. Given a liberal strategy ς and a state $s \in S$, we define the *importance* of s by

$$Imp^{\varsigma}(s) := \mathbb{P}^{\varsigma}[\Diamond s \mid \Diamond F]$$

the probability of visiting s conditioned on reaching F (afterwards). Intuitively, the importance is high for states where a good decision can help to reach the target.[6]

Example 1. For the MDP of Fig. 1 with the objective $\Diamond\{t\}$ and a strategy ς choosing b, we have $Imp^{\varsigma}(s) = 1$ and $Imp^{\varsigma}(q) = 5/995$. Trivially, $Imp^{\varsigma}(t) = 1$. For all other states, the importance is zero.

Obviously, decisions made in states of zero importance do not affect $\mathbb{P}^{\varsigma}[\Diamond F]$ since these states never occur on paths from $\hat{s}$ to F. However, note that many states of S may be reachable in G^{ς} with positive but negligibly small probability. Clearly, the value of $\mathbb{P}^{\varsigma}[\Diamond F]$ depends only marginally on choices made in these states. Formally, let ς_{Δ} be a strategy obtained from ς by changing each $\varsigma(s)$ with $Imp^{\varsigma}(s) \leq \Delta$ to an arbitrary subset of $Act(s)$. We obtain the following obvious property:

Lemma 2. *For every liberal strategy ς, we have* $\lim_{\Delta \to 0} \mathbb{P}^{\varsigma_{\Delta}}[\Diamond F] = \mathbb{P}^{\varsigma}[\Diamond F]$.

In fact, every $\Delta < \min(\{Imp^{\varsigma}(s) \mid s \in S\} \setminus \{0\})$ satisfies $\mathbb{P}^{\varsigma_{\Delta}}[\Diamond F] = \mathbb{P}^{\varsigma}[\Diamond F]$. But often even larger Δ may give $\mathbb{P}^{\varsigma_{\Delta}}[\Diamond F]$ sufficiently close to $\mathbb{P}^{\varsigma}[\Diamond F]$. Such Δ may be found using e.g. trial and error approach.[7]

Most importantly, we can use the importance of a state to affect the probability that decisions in this state are indeed remembered in the data structure. Data structures with such a feature are used in various learning algorithms. In the next section, we discuss decision trees. Due to this extra variability, which decisions to learn, the resulting decision trees are smaller than BDDs for strictly defined ς_{Δ}.

5 Efficient Representations

Let $G = (S, A, Act, \delta, \hat{s})$ be an MDP. In order to symbolically represent strategies in G, we need to assume that states and actions have some internal structure. Inspired by PRISM language [KNP11], we consider a set $\mathcal{V} = \{v_1, \ldots, v_n\}$ of *integer variables*, each v_i gets its values from a finite domain $Dom(v_i)$. We suppose that $S = \prod_{i=1}^{n} Dom(v_i) \subseteq \mathbb{Z}^n$, i.e. each state is a vector of integers. Further, we assume that the MDP arises as a product of m modules, each of which can separately perform non-synchronizing actions as well as synchronously with other modules perform a synchronizing action. Therefore, we suppose $A \subseteq \bar{A} \times \{0, \ldots, m\}$, where $\bar{A} \subseteq \mathbb{N}$ is a finite

[6] Instead of the conditional probability of reaching s, we could consider the conditional expected number of visits of s. We discuss the differences and compare the efficiency together with the case of no conditioning on reaching the target in Section 6.

[7] One may give a theoretical bound on convergence of $\mathbb{P}^{\varsigma_{\Delta}}[\Diamond F]$ to $\mathbb{P}^{\varsigma}[\Diamond F]$ as $\Delta \to 0$, using e.g. Lemma 5.1 of [BKK14]. However, for large MDPs the bound would be impractical.

set and the second component determines the module performing the action (0 stands for synchronizing actions).[8]

Since a liberal strategy is a function of the form $\varsigma : S \to 2^A$, assigning to each state its good actions, it can be *explicitly* represented as a list of state-action pairs, i.e., as a subset of

$$S \times A = \prod_{i=1}^{n} Dom(v_i) \times \bar{A} \times \{0, 1, \dots, m\} \tag{1}$$

In addition, standard optimization algorithms implemented in PRISM use an explicit "don't-care" value -2 for action in each unreachable state, meaning the strategy is not defined. However, one could simply not list these pairs at all. Thus a smaller list is obtained, with only the states where ς is defined. Recall that one may also omit states s satisfying $\mathit{Imp}^{\varsigma}(s) = 0$, thus ignoring reachable states with zero probability to reach the target. Further optimization may be achieved by omitting states s satisfying $\mathit{Imp}^{\varsigma}(s) < \varDelta$ for a suitable $\varDelta > 0$.

5.1 BDD Representation

The explicit set representation can be encoded as a binary decision diagram (BDD). This has been used in e.g. [WBB$^+$10,EJPV12]. The principle of the BDD representation of a set is that (1) each element is encoded as a string of bits and (2) an automaton, in the form of a binary directed acyclic graph, is created so that (3) the accepted language is exactly the set of the given bit strings. Although BDDs are quite efficient, see Section 6, each of these three steps can be significantly improved:

1. Instead of a string of bits describing all variables, a string of integers (one per variable) can be used. Branching is then done not on the value of each bit, but according to an inequality comparing the variable to a constant. This significantly improves the readability.
2. Instead of building the automaton according to a chosen order of bits, we let a heuristic choose the order of the inequalities and the actual constants in the inequalities.
3. Instead of representing the language precisely, we allow the heuristic to choose which data to represent and which not. The likelihood that each datum is represented corresponds to its importance, which we provide as another input.

The latter two steps lead to significantly smaller graphs than BDDs. All this can be done in an efficient way using decision trees learning.

5.2 Representation using Decision Trees

Decision trees. A *decision tree* for a domain $\prod_{i=1}^{d} X_i \subseteq \mathbb{Z}^d$ is a tuple $\mathcal{T} = (T, \rho, \theta)$ where T is a finite rooted binary (ordered) tree with a set of inner nodes N and a set of leaves L, ρ assigns to every inner node a predicate of the form $[x_i \sim \mathit{const}]$ where $i \in \{1, \dots, d\}$, $x_i \in X_i$, $\mathit{const} \in \mathbb{Z}$, $\sim \in \{\leq, <, \geq, >, =\}$, and θ assigns to every leaf a value *good*, or *bad*. [9]

[8] On the one hand, PRISM does not allow different modules to have local variables with the same name, hence we do not distinguish which module does a variable belong to. On the other hand, while PRISM declares no names for non-synchronizing actions, we want to exploit the connection between the corresponding actions of different copies of the same module.

[9] There exist many variants of decision trees in the literature allowing arbitrary branching, arbitrary values in the leaves, etc., e.g. [Mit97]. For simplicity, we define only a special suitable subclass.

Similarly to BDDs, the language $\mathcal{L}(\mathcal{T}) \subseteq \mathbb{N}^n$ of the tree is defined as follows. For a vector $\bar{\boldsymbol{x}} = (\bar{x}_1, \dots, \bar{x}_n) \in \mathbb{N}^n$, we find a path p from the root to a leaf such that for each inner node n on the path, the predicate $\rho(n)$ is satisfied by substitution $x_i = \bar{x}_i$ iff the first child of n is on p. Denote the leaf on this particular path by ℓ. Then $\bar{\boldsymbol{x}}$ is in the language $\mathcal{L}(\mathcal{T})$ of $\mathcal{T}$ iff $\theta(\ell) = \mathit{good}$.

Example 2. Consider dimension $d = 1$, domain $X_1 = \{1, \dots, 7\}$. A tree representing a set $\{1, 2, 3, 7\}$ is depicted in Fig. 2. To depict the ordered tree clearly, we use unbroken lines for the first child, corresponding to the satisfied predicate, and dashed line for the second one, corresponding to the unsatisfied predicate.

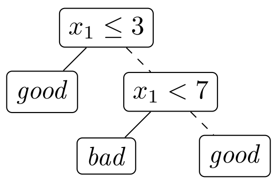


Fig. 2: A decision tree for $\{1, 2, 3, 7\} \subseteq \{1, \dots, 7\}$

In our setting, we use the domain $S \times A$ defined by Equation (1) which is of the form $\prod_{i=1}^{n+2} X_i$ where for each $1 \leq i \leq n$ we have $X_i = \mathit{Dom}(v_i)$, $X_{n+1} = \bar{A}$ and $X_{n+2} = \{0, 1, \dots, m\}$. Here the coordinates $\mathit{Dom}(v_i)$ are considered "unbounded" and, consequently, the respective predicates use inequalities. In contrast, we know the possible values of $\bar{A} \times \{0, 1, \dots, m\}$ in advance and they are not too many. Therefore, these coordinates are considered "discrete" and the respective predicates use equality. Examples of such trees are given in Section 6 in Fig. 4 and 5. Now a decision tree $\mathcal{T}$ for this domain determines a liberal strategy $\varsigma : S \to 2^A$ by $a \in \varsigma(s)$ iff $(s, a) \in \mathcal{L}(\mathcal{T})$.

Learning. We describe the process of *learning a training set*, which can also be understood as storing the input data. Given a training sequence (repetitions allowed!) $\boldsymbol{x}^1, \dots, \boldsymbol{x}^k$, with each $\boldsymbol{x}^i = (x^i_1, \dots, x^i_n) \in \mathbb{N}^d$, partitioned into the *positive* and *negative* subsequence, the process of learning according to the algorithm ID3 [Qui86,Mit97] proceeds as follows:

1. Start with a single node (root), and assign to it the whole training sequence.
2. Given a node n with a sequence τ,
   – if all training examples in τ are positive, set $\theta(n) = \mathit{good}$ and stop;
   – if all training examples in τ are negative, set $\theta(n) = \mathit{bad}$ and stop;
   – otherwise,
      • choose a predicate with the "highest gain" (with lowest entropy, see e.g. [Mit97, Sections 3.4.1, 3.7.2]),
      • split τ into sequences satisfying and not satisfying the predicate, assign them to the first and the second child, respectively,
      • go to step 2 for each child.

Intuitively, the predicate with the highest gain splits the sequence so that it maximizes the portion of positive data in the satisfying subsequence and the portion of negative data in the non-satisfying subsequence.

In addition, the final tree can be *pruned*. This means that some leaves are merged, resulting in a smaller tree at the cost of some imprecision of storing (the language of the tree changes). The pruning phase is quite sophisticated, hence for the sake of simplicity and brevity, we omit the details here. We use the standard C4.5 algorithm and refer to [Qui93,Mit97]. In Section 6, we comment on effects of parameters used in pruning.

Learning a strategy. Assume that we already have a liberal strategy $\varsigma : S \to 2^A$. We show how we learn good and bad state-action pairs so that the language of the resulting tree is close to the set of good pairs. The training sequence will be composed of state-action pairs where good pairs are positive examples, and bad pairs are negative ones. Since our aim is to ensure that important states are learnt and not pruned away, we repeat pairs with more important states in the training sequence more frequently.

Formally, for every $s \in S$ and $a \in \mathit{Act}(s)$, we put the pair (s, a) to the training sequence $\mathit{repeat}(s)$-times, where

$$\mathit{repeat}(s) = c \cdot \mathit{Imp}^{\varsigma}(s)$$

for some constant $c \in \mathbb{N}$ (note that $\mathit{Imp}^{\varsigma}(s) \leq 1$). Since we want to avoid exact computation of $\mathit{Imp}^{\varsigma}(s)$, we estimate it using simulations. In practice, we thus run c simulation runs that reach the target and set $\mathit{repeat}(s)$ to be the number of runs where s was also reached.

6 Experiments

In this section, we present the experimental evaluation of the presented methods, which we have implemented within the probabilistic model checker PRISM [KNP11]. All the results presented in this section were obtained on a single Intel(R) Xeon(R) CPU (3.50GHz) with memory limited to 10GB.

First, we discuss several alternative options to construct the training data and to learn them in a decision tree. Further, we compare decision trees to other data structures, namely sets and BDDs, with respect to the sizes necessary for storing a strategy. Finally, we illustrate how the decision trees can be used to gain insight into our benchmarks.

6.1 Decision Tree Learning

Generating Training Data. The strategies we work with come from two different sources. Firstly, we consider strategies constructed by PRISM, which can be generated using the explicit or sparse model checking engine. Secondly, we consider strategies constructed by the BRTDP algorithm [BCC^{+}14], which are defined on a part of the state space only.

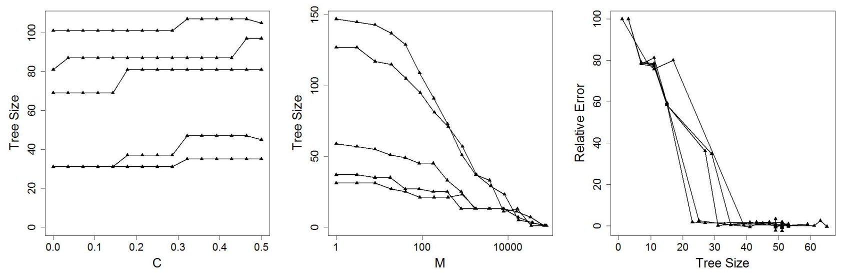
Recall that given a strategy, the training data for the decision trees is constructed from c simulation runs according to the strategy. In our experiments, we found that $c = 10000$ produces good results in all the examples we consider. Note that we stop each simulation as soon as the target or a state with no path to the target state is reached.

Decision Tree Learning in Weka. The decision trees are constructed using the Weka machine learning package [HFH^{+}09]. The Weka suite offers various decision tree classifiers. We use the J48 classifier, which is an implementation of the C4.5 algorithm [Qui93]. The J48 classifier offers two parameters to control the pruning that affect the size of the decision tree:

- The leaf size parameter $M \in \mathbb{N}$ determines that each leaf node with less than M instances in the training data is merged with its siblings. Therefore, only values smaller than the number of instances per classification class are reasonable, since higher numbers always result in the trivial tree of size 1.
- The confidence factor $C \in (0, 0.5]$ is used internally for determining the amount of pruning during decision tree construction. Smaller values incur more pruning and therefore smaller trees.

Detailed information and an empirical study of the parameters for J48 is available in [DM12].

Effects of the parameters. We illustrate the effects of the parameters C and M on the resulting size of the decision tree on the `mer` benchmark. However, similar behaviour appears in all the examples. Figures 3a and 3b show the resulting size of the decision tree for several (random) executions. Each line in the plots corresponds to one decision tree, learned with 15 different values of the parameter. The C parameter scales linearly between 0.0001 and 0.5. The M parameter scales logarithmically between 1 and the minimum number of instances per class in the respective training set. The plots in Figure 3 show that M is an effective parameter in calibrating the resulting tree size, whereas C plays less of a role. Hence, we use $C = 10^{-4}$. Furthermore, since the tree size is monotone in M, the parameter M can be used to retrieve a desired level of detail.



(a) fixed $M = 2$ (b) fixed $C = 10^{-4}$ (c) Tree Size vs Error

Fig. 3: Decision tree Parameters

Figure 3c depicts the relation of the tree size to the relative error of the induced strategy. It shows that there is a threshold size under which the tree is not able to capture the strategy correctly anymore and the error rises quickly. Above the threshold size, the error is around 1%, considered reasonable in order to extract reliable information. This threshold behaviour is observed in all our examples. Therefore, it is sensible to perform a binary search for the highest M ensuring the error at most 1%.

6.2 Results

First, we briefly introduce the four examples from the PRISM benchmark suite [KNP12], which we tested our method on. Note that the majority of the states in the used benchamrks are non-deterministic, so the strategies are non-trivial in most states.

firewire models the Tree Identify Protocol of the IEEE 1394 High Performance Serial Bus, which is used to transport video and audio signals within a network of multimedia devices. The reachability objective is that one node gets the root and the other one the child role.

investor models a market investor and shares, which change their value probabilistically over time. The reachability objective is that the investor finishes a trade at a time, when his shares are more valuable than some threshold value.

Table 2: Effects of various learning variants on the tree size. Smallest trees computed from PRISM or BRTDP and the average time to compute one number are presented.

Example	I$\Diamond\mathbb{P}$	I$\forall\mathbb{P}$	I$\Diamond\mathbb{E}$	I$\forall\mathbb{E}$	O$\Diamond$	O$\forall$	Avg Time
firewire	1	1	1	1	1	1	45s
investor	27	25	31	35	37	37	135s
mer_17M	17	33	17	29	19	none	314s
mer_big	19	23	23	37	17	none	129s
zeroconf	7	7	7	7	7	17	141s

Table 1: Comparison of representation sizes for strategies obtained from PRISM and from BRTDP computation. Sizes are presented as the number of states for explicit lists of values, the number of nodes for BDDs, and the number of nodes for decision trees (DT). For DT, we display the tree size obtained from the binary search described above. DT Error reports the relative error of the strategy determined by the decision tree (on the induced Markov chain) compared to the optimal value, obtained by model checking with PRISM.

			PRISM				**BRTDP**			
Example	$\|S\|$	Value	Explicit	BDD	DT	DT Error	Explicit	BDD	DT	DT Error
firewire	481,136	1.000	479,834	4,233	1	0.000%	766	4,763 [1]	1	0.000%
investor	35,893	0.958	28,151	783	27	0.886%	21,931	2,780	35	0.836%
mer_17M	1,773,664	0.200	Memory Out				1,887	619	17	0.000%
mer_big [2]	Approx. 10^{13}	0.200	Memory Out				1,894	646	19	0.692%
zeroconf	89,586	0.009	60,463	409	7	0.106%	1,630	905	7	0.235%

[1] Note that BDDs represent states in binary form. Therefore, one entry in the explicit state list corresponds to several nodes in the BDD.
[2] We did not measure the state size as the MDP does not fit in memory, but extrapolated it from the linear dependence of model size and one of its parameters, which we could increase to $2^{31} - 1$. The value is obtained from the BRTDP computation.

mer is a mutual exclusion protocol, that regulates the access of two users to two different resources. The protocol should prohibit that both resources are accessed simultaneously. **zeroconf** is a network protocol which allows users to choose their IP addresses autonomously. The protocol should detected and prohibit IP address conflict.

For every example, Table 1 shows the size of the state space, the value of the optimal strategy, and the sizes of strategies obtained from explicit model checking by PRISM and by BRTDP, for each discussed data structure.

Learning variants. In order to justify our choice of the importance function Imp, we compare it to several alternatives.

1. When constructing the training data, we can use the importance measure Imp, and add states as often as is indicated by its importance (I), or neglect it and simply add every visited state exactly once (O).
2. Further, states on the simulation are learned conditioned on the fact that the target state is reached ($\Diamond$). Another option is to consider all simulations ($\forall$).
3. Finally, instead of the probability to visit the state ($\mathbb{P}$), one can consider the expected number of visits ($\mathbb{E}$).

In Table 2, we report the sizes of the decision trees obtained for the all learning variants. We conclude that our choice (I$\Diamond\mathbb{P}$) is the most useful one.

6.3 Understanding Decision Trees

We show how the constructed decision trees can help us to gain insight into the essential features of the systems.

`zeroconf` example. In Figure 4 we present a decision tree that is a strategy for `zeroconf` and shows how an unresolved IP address conflict can occur in the protocol. First we present how to read the strategy represented in Figure 4. Next we show how the strategy can explain the conflict in the protocol. Assume that we are classifying a state-action pair (s, a), where action a is enabled in state s.

1. No matter what the current state s is, the action `rec` is always classified as bad according to the root of the tree. Therefore, the action `rec` should be played with positive probability only if all other available actions in the current state are also classified as bad.
2. If action a is different from `rec`, the right son of the root node is reached. If action a is different from action `l>0&b=1&ip_mess=1 -> b'=0&z'=0&n1'=min(n1+1,8)&ip_mess'=0` (the whole PRISM command is a single action), then a is classified as $good$ in state s. Otherwise, the left son is reached.
3. In node $\mathtt{z} \le 0$ the classification of action a (that is the action that labels the parent node) depends on the variable valuation of the current state. If the value of var. z is greater than 0, then a is classified as $good$ in state s, otherwise it is classified as bad.

Action `rec` stands for a network host receiving a reply to a broadcast message, resulting in resolution of an IP address conflict if one is present, which clearly does not help in constructing an unresolved conflict. The action labelling the right son of the root represents the detection of an IP address conflict by an arbitrary network host. This action is only good, if variable z, which is a clock variable, in the current state is greater than 0. The combined meaning of the two nodes is that an unresolved IP address conflict can occur if the conflict is detected too late.

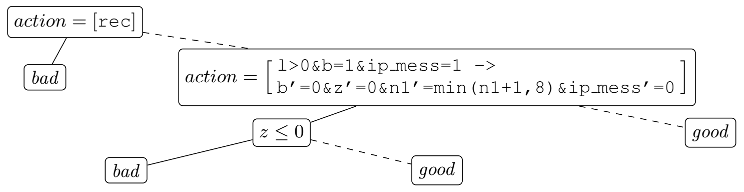

$action = [\mathtt{rec}]$
bad
$action = \left[\begin{array}{l}\mathtt{l>0\&b=1\&ip\_mess=1\ ->} \\ \mathtt{b'=0\&z'=0\&n1'=min(n1+1,8)\&ip\_mess'=0}\end{array}\right]$
$z \le 0$
$good$
bad
$good$

Fig. 4: A decision tree for `zeroconf`

`firewire` example. For `firewire`, we obtain a trivial tree with a single node, labelled $good$. Therefore, playing all available actions in each state guarantees reaching the target almost surely. In contrast to other representations, we have automatically obtained the information that the network always reaches the target configuration, regardless of the individual behaviour of its components and their interleaving.

`mer` example. In the case of `mer`, there exists a strategy that violates the required property that the two resources are not accessed simultaneously. The decision tree for the `mer` strategy is depicted in Figure 5. In order to understand how a state is reached, where both resources are accessed at the same time, it is necessary to determine which user accesses which resource in that state.

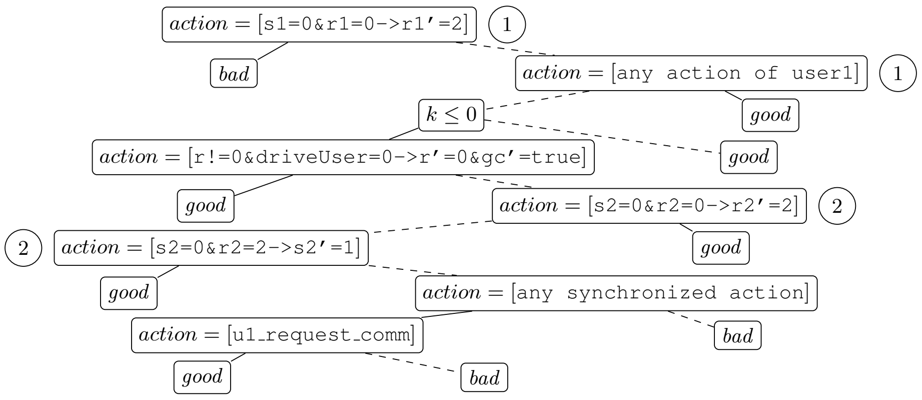
1. The two tree nodes labelled by 1 explain what resource *user 1* should access. The root node labelled by action `s1=0&r1=0 -> r1'=2` specifies that the request to access *resource* 2 (variable `r1` is set to 2) is classified as bad. The only remaining action for *user 1* is to request access to *resource* 1. This action is classified as $good$ by the right son of the root node.
2. Analogously, the tree nodes labelled by 2 specify that *user 2* actions should request access to *resource 2* (follows from `s2=0&r2=0 -> r2'=2`). Once *resource 2* is requested it should change its internal state `s2` to 1 (follows from `s2=0&r2=2`

`-> s2'=1`). It follows, that in the state violating the property, *user 1* has access to *resource 1* and *user 2* to *resource 2*.

The model is supposed to correctly handle such overlapping requests, but fails to do so in a specific case. In order to further debug the model, one has to find the action of the scheduler that causes this undesired behaviour. The lower part of the tree specifies that `u1_request_comm` is a candidate for such an action. Inspecting a snippet of the code of `u1_request_comm` from the PRISM source code (shown below) reveals that in the given situation, the scheduler reacts inappropriately with some probability p.

```
[u1_request_comm]  s=0 & commUser=0 & driveUser!=0 & k<n ->
           (1-p):(s'=1) & (r'=driveUser) & (k'=k+1) +
               p:(s'=-1) & (gc'=true) & (k'=k+1)
```

The remaining nodes of the tree that were not discussed are necessary to reset the situation if the non-faulty part (with probability $1-p$) of the `u1_request_comm` command was executed. It should be noted that executing the faulty `u1_request_comm` action does not lead to the undesired state right away. The action only grants *user 1* access rights in a situation, where he should not get these rights. Only a successive action leads to *user 1* accessing the resource and the undesired state being reached. This is a common type of bug, where the command that triggered an error is not the cause of it.



action = [s1=0&r1=0->r1'=2]
1
bad
action = [any action of user1]
1
k ≤ 0
good
action = [r!=0&driveUser=0->r'=0&gc'=true]
good
good
action = [s2=0&r2=0->r2'=2]
2
2
action = [s2=0&r2=2->s2'=1]
good
good
action = [any synchronized action]
action = [u1_request_comm]
bad
good
bad

Fig. 5: A decision tree for `mer`

7 Conclusion

In this work we presented a new approach to represent strategies in MDPs in a succinct and comprehensible way. We exploited machine learning methods to achieve our goals. Interesting directions of future works are to investigate whether other machine learning methods can be integrated with our approach, and to extend our approach from reachability objectives to other objectives (such as long-run average and discounted-sum).

Acknowledgements. This research was funded in part by Austrian Science Fund (FWF) Grant No P 23499-N23, FWF NFN Grant No S11407-N23 (RiSE) and Z211-N23 (Wittgenstein Award), European Research Council (ERC) Grant No 279307 (Graph Games), ERC Grant No 267989 (QUAREM), the Czech Science Foundation Grant No P202/12/G061, and People Programme (Marie Curie Actions) of the European Union's Seventh Framework Programme (FP7/2007-2013) REA Grant No 291734.

References

ÁBD$^+$14. E. Ábrahám, B. Becker, C. Dehnert, N. Jansen, J.-P. Katoen, and R. Wimmer. Counterexample generation for discrete-time markov models: An introductory survey. In *Formal Methods for Executable Software Models - 14th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2014, Bertinoro, Italy, June 16-20, 2014*, pages 65–121, 2014.

ADvR08. M. E. Andrés, P. R. D'Argenio, and P. van Rossum. Significant diagnostic counterexamples in probabilistic model checking. In *Hardware and Software: Verification and Testing, 4th International Haifa Verification Conference, HVC 2008, Haifa, Israel, October 27-30, 2008. Proceedings*, pages 129–148, 2008.

AL09. Husain Aljazzar and Stefan Leue. Generation of counterexamples for model checking of markov decision processes. In *QEST*, pages 197–206. IEEE Computer Society, 2009.

AL10. Husain Aljazzar and Stefan Leue. Directed explicit state-space search in the generation of counterexamples for stochastic model checking. *IEEE Trans. Software Eng.*, 36(1):37–60, 2010.

ALFLS11. H. Aljazzar, F. Leitner-Fischer, S. Leue, and D. Simeonov. Dipro - a tool for probabilistic counterexample generation. In *SPIN*, pages 183–187, 2011.

BCC$^+$14. Tomáš Brázdil, Krishnendu Chatterjee, Martin Chmelík, Vojtěch Forejt, Jan Křetínský, Marta Z. Kwiatkowska, David Parker, and Mateusz Ujma. Verification of markov decision processes using learning algorithms. In *ATVA 2014, Sydney, NSW, Australia, November 3-7, 2014, Proceedings*, pages 98–114, 2014.

BD96. C. Boutilier and R. Dearden. Approximating value trees in structured dynamic programming. In *In Proceedings of the Thirteenth International Conference on Machine Learning*, pages 54–62, 1996.

BDG95. C. Boutilier, R. Dearden, and M. Goldszmidt. Exploiting structure in policy construction. In *IJCAI-95, pp.11041111*, 1995.

BDH99. C. Boutilier, T. Dean, and S. Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *JAIR*, 11:1–94, 1999.

BH11. Tevfik Bultan and Pao-Ann Hsiung, editors. *Automated Technology for Verification and Analysis, 9th International Symposium, ATVA, Taipei, Taiwan, October 11-14, 2011. Proceedings*, volume 6996 of *LNCS*. Springer, 2011.

BJW02. Julien Bernet, David Janin, and Igor Walukiewicz. Permissive strategies: from parity games to safety games. *ITA*, 36(3):261–275, 2002.

BK08. C. Baier and J.-P. Katoen. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.

BKK14. T. Brázdil, S. Kiefer, and A. Kučera. Efficient analysis of probabilistic programs with an unbounded counter. *Journal of the ACM*, 61(6):41:1–41:35, 2014.

BMOU11. Patricia Bouyer, Nicolas Markey, Jörg Olschewski, and Michael Ummels. Measuring permissiveness in parity games: Mean-payoff parity games revisited. In Bultan and Hsiung [BH11], pages 135–149.

CCD14. Krishnendu Chatterjee, Martin Chmelik, and Przemyslaw Daca. CEGAR for qualitative analysis of probabilistic systems. In *CAV*, pages 473–490, 2014.

CK91. D. Chapman and L. P. Kaelbling. Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. pages 726–731. Morgan Kaufmann, 1991.

CV10. Rohit Chadha and Mahesh Viswanathan. A counterexample-guided abstraction-refinement framework for Markov decision processes. *ACM Trans. Comput. Log. 12*, page 1, 2010.

CY95. C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *Journal of the ACM*, 42(4):857–907, 1995.

dA97. L. de Alfaro. *Formal Verification of Probabilistic Systems*. PhD thesis, Stanford University, 1997.

dAKN$^+$00. L. de Alfaro, M. Kwiatkowska, G. Norman, D. Parker, and R. Segala. Symbolic model checking of probabilistic processes using MTBDDs and the Kronecker representation. In *TACAS*, 2000.

DFK$^+$14. Klaus Dräger, Vojtech Forejt, Marta Z. Kwiatkowska, David Parker, and Mateusz Ujma. Permissive controller synthesis for probabilistic systems. In *TACAS*, pages 531–546, 2014.

DHK08. B. Damman, T. Han, and J.-P. Katoen. Regular expressions for PCTL counterexamples. In *QEST*, pages 179–188. IEEE Computer Society, 2008.

DJJL01. Pedro R. D'Argenio, Bertrand Jeannet, Henrik Ejersbo Jensen, and Kim Guldstrand Larsen. Reachability analysis of probabilistic systems by successive refinements. In *PAPM-PROBMIV*, LNCS 2165, pages 39–56. Springer, 2001.

DJJL02. Pedro R. D'Argenio, Bertrand Jeannet, Henrik Ejersbo Jensen, and Kim Guldstrand Larsen. Reduction and refinement strategies for probabilistic analysis. In *PAPM-PROBMIV*, LNCS 2399, pages 57–76. Springer, 2002.

DJW$^+$14. C. Dehnert, N. Jansen, R. Wimmer, E. Ábrahám, and J.-P. Katoen. Fast debugging of PRISM models. In Franck Cassez and Jean-François Raskin, editors, *Automated Technology for Verification and Analysis - 12th International Symposium, ATVA 2014, Sydney, NSW, Australia, November 3-7, 2014, Proceedings*, volume 8837 of *Lecture Notes in Computer Science*, pages 146–162. Springer, 2014.

DM12. Sam Drazin and Matt Montag. Decision tree analysis using weka. *Machine Learning-Project II, University of Miami*, pages 1–3, 2012.

EJPV12. C. Von Essen, B. Jobstmann, D. Parker, and R. Varshneya. Semi-symbolic computation of efficient controllers in probabilistic environments. Technical report, Verimag, 2012.

FHPW10. H. Fecher, M. Huth, N. Piterman, and D. Wagner. PCTL model checking of markov chains: Truth and falsity as winning strategies in games. *Perform. Eval.*, 67(9):858–872, 2010.

FV97. J. Filar and K. Vrieze. *Competitive Markov Decision Processes*. Springer-Verlag, 1997.

HFH$^+$09. Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

HKD09. T. Han, J.-P. Katoen, and B. Damman. Counterexample generation in probabilistic model checking. *IEEE Trans. Software Eng.*, 35(2):241–257, 2009.

HKN$^+$03. H. Hermanns, M. Kwiatkowska, G. Norman, D. Parker, and M. Siegle. On the use of MTBDDs for performability analysis and verification of stochastic systems. *Journal of Logic and Algebraic Programming: Special Issue on Probabilistic Techniques for the Design and Analysis of Systems*, 56(1-2):23–67, 2003.

How60. Ronald A. Howard. *Dynamic programming and Markov processes*. The MIT press, New York London, Cambridge, MA, 1960.

HSaHB99. J. Hoey, R. St-aubin, A. Hu, and C. Boutilier. Spudd: Stochastic planning using decision diagrams. In *In Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pages 279–288. Morgan Kaufmann, 1999.

HWZ08. H. Hermanns, B. Wachter, and L. Zhang. Probabilistic CEGAR. In *CAV*, pages 162–175, 2008.

JÁK$^+$11. N. Jansen, E. Ábrahám, J. Katelaan, R. Wimmer, J.-P. Katoen, and B. Becker. Hierarchical counterexamples for discrete-time markov chains. In Bultan and Hsiung [BH11], pages 443–452.

JÁV$^+$12. N. Jansen, E. Ábrahám, M. Volk, R. Wimmer, J.-P. Katoen, and B. Becker. The COMICS tool - computing minimal counterexamples for dtmcs. In *ATVA*, pages 349–353, 2012.

KH09. M. Kattenbelt and M. Huth. Verification and refutation of probabilistic specifications via games. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2009, December 15-17, 2009, IIT Kanpur, India*, pages 251–262, 2009.

KHW94. N. Kushmerick, S. Hanks, and D. Weld. An algorithm for probabilistic least-commitment planning. In *Proceedings of AAAI-94*, page 10731078, 1994.

KK99. M. Kearns and D. Koller. Efficient reinforcement learning in factored MDPs. IJCAI, pages 740–747, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

KNP06. Marta Z. Kwiatkowska, Gethin Norman, and David Parker. Game-based abstraction for Markov decision processes. In *QEST*, pages 157–166, 2006.

KNP11. M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *CAV*, pages 585–591, 2011.

KNP12. M. Kwiatkowska, G. Norman, and D. Parker. The PRISM benchmark suite. In *QEST*, pages 203–204, 2012.

KP99. D. Koller and R. Parr. Computing factored value functions for policies in structured MDPs. In *In Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 1332–1339. Morgan Kaufmann, 1999.

KP13. Marta Kwiatkowska and David Parker. Automated verification and strategy synthesis for probabilistic systems. In *Automated Technology for Verification and Analysis*, pages 5–22. Springer, 2013.

KPC12. A. Komuravelli, C. S. Pasareanu, and E. M. Clarke. Assume-guarantee abstraction refinement for probabilistic systems. In *CAV*, pages 310–326, 2012.

LL13. F. Leitner-Fischer and S. Leue. Probabilistic fault tree synthesis using causality computation. *IJCCBS*, 4(2):119–143, 2013.

Mit97. Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

MLG05. H Brendan McMahan, Maxim Likhachev, and Geoffrey J Gordon. Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. In *ICML*, 2005.

MP04. A. Miner and D. Parker. *Validation of Stochastic Systems: A Guide to Current Research*, volume 2925 of *Lecture Notes in Computer Science (Tutorial Volume)*, chapter Symbolic Representations and Analysis of Large Probabilistic Systems, pages 296–338. Springer, 2004.

Put94. M.L. Puterman. *Markov Decision Processes*. Wiley, 1994.

Pye03. Larry D. Pyeatt. Reinforcement learning with decision trees. In *The 21st IASTED International Multi-Conference on Applied Informatics (AI 2003), February 10-13, 2003, Innsbruck, Austria*, pages 26–31, 2003.

Qui86. J. Ross Quinlan. Induction of decision trees. *Mach. Learn.*, 1(1):81–106, 1986.

Qui93. J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

Seg95. R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems*. PhD thesis, MIT Press, 1995. Technical Report MIT/LCS/TR-676.

SLT10. C. S. Raghavendra S. Liu, A. Panangadan and A. Talukder. Compact representation of coordinated sampling policies for body sensor networks. In *In Proceedings of Workshop on Advances in Communication and Networks (Smart Homes for Tele-Health)*, pages 6–10. IEEE, 2010.

Var85. M. Vardi. Automatic verification of probabilistic concurrent finite state programs. In *FOCS*, pages 327–338, 1985.

WBB$^+$10. Ralf Wimmer, Bettina Braitling, Bernd Becker, Ernst Moritz Hahn, Pepijn Crouzen, Holger Hermanns, Abhishek Dhama, and Oliver Theel. Symblicit calculation of long-run averages for concurrent probabilistic systems. In *QEST*, pages 27–36, Washington, DC, USA, 2010. IEEE Computer Society.

WJÁ+14. R. Wimmer, N. Jansen, E. Ábrahám, J.-P. Katoen, and B. Becker. Minimal counterexamples for linear-time probabilistic verification. *TCS*, 549:61–100, 2014.

WJV+13. Ralf Wimmer, Nils Jansen, Andreas Vorpahl, Erika Ábrahám, Joost-Pieter Katoen, and Bernd Becker. High-level counterexamples for probabilistic automata. In *QEST*, pages 39–54, 2013.

UNIFYING TWO VIEWS ON MULTIPLE MEAN-PAYOFF OBJECTIVES IN MARKOV DECISION PROCESSES

KRISHNENDU CHATTERJEE, ZUZANA KŘETÍNSKÁ, AND JAN KŘETÍNSKÝ

IST Austria

Institut für Informatik, Technische Universität Muñchen, Germany

Institut für Informatik, Technische Universität Muñchen, Germany

Abstract. We consider Markov decision processes (MDPs) with multiple limit-average (or mean-payoff) objectives. There exist two different views: (i) the expectation semantics, where the goal is to optimize the expected mean-payoff objective, and (ii) the satisfaction semantics, where the goal is to maximize the probability of runs such that the mean-payoff value stays above a given vector. We consider optimization with respect to both objectives at once, thus unifying the existing semantics. Precisely, the goal is to optimize the expectation while ensuring the satisfaction constraint. Our problem captures the notion of optimization with respect to strategies that are risk-averse (i.e., ensure certain probabilistic guarantee). Our main results are as follows: First, we present algorithms for the decision problems, which are always polynomial in the size of the MDP. We also show that an approximation of the Pareto curve can be computed in time polynomial in the size of the MDP, and the approximation factor, but exponential in the number of dimensions. Second, we present a complete characterization of the strategy complexity (in terms of memory bounds and randomization) required to solve our problem.

1. Introduction

MDPs and mean-payoff objectives. The standard models for dynamic stochastic systems with both nondeterministic and probabilistic behaviours are Markov decision processes (MDPs) [How60, Put94, FV97]. An MDP consists of a finite state space, and in every state a controller can choose among several actions (the nondeterministic choices), and given the current state and the chosen action the system evolves stochastically according to a probabilistic transition function. Every action in an MDP is associated with a reward (or cost), and the basic problem is to obtain a strategy (or policy) that resolves the choice of actions in order to optimize the rewards obtained over the run of the system. An objective is a

This is an extended version of the LICS'15 paper with full proofs and additional complexity results. This research was funded in part by Austrian Science Fund Grant No P 23499-N23, European Research Council Grant No 279307 (Graph Games), the DFG Research Training Group PUMA: Programm- und Modell-Analyse (GRK 1480), the Czech Science Foundation grant No. 15-17564S, and People Programme (Marie Curie Actions) of the European Union's Seventh Framework Programme (FP7/2007-2013) REA Grant No 291734.

LOGICAL METHODS
IN COMPUTER SCIENCE

DOI:10.2168/LMCS-???

function that given a sequence of rewards over the run of the system combines them to a single value. A classical and one of the most well-studied objectives in context of MDPs is the *limit-average (or long-run average or mean-payoff)* objective that assigns to every run the average of the rewards over the run.

Single vs. multiple objectives. MDPs with single mean-payoff objectives have been widely studied (see, e.g., [Put94, FV97]), with many applications ranging from computational biology to analysis of security protocols, randomized algorithms, or robot planning, to name a few [BK08, KNP02, DEKM98, KGFP09]. In verification of probabilistic systems, MDPs are widely used, for concurrent probabilistic systems [CY95, Var85], probabilistic systems operating in open environments [Seg95, dA97], and applied in diverse domains [BK08, KNP02]. However, in several application domains, there is not a single optimization goal, but multiple, potentially dependent and conflicting goals. For example, in designing a computer system, the goal is to maximize average performance while minimizing average power consumption, or in an inventory management system, the goal is to optimize several potentially dependent costs for maintaining each kind of product. These motivate the study of MDPs with multiple mean-payoff objectives, which has also been applied in several problems such as dynamic power management [FKP12].

Two views. There exist two views in the study of MDPs with mean-payoff objectives [BBC+14]. The traditional and classical view is the *expectation* semantics, where the goal is to maximize (or minimize) the expectation of the mean-payoff objective. There are numerous applications of MDPs with the expectation semantics, such as in inventory control, planning, and performance evaluation [Put94, FV97]. The alternative semantics is called the *satisfaction* semantics, which, given a mean-payoff value threshold sat and a probability threshold pr, asks for a strategy to ensure that the mean-payoff value be at least sat with probability at least pr. In the case with n reward functions, there are two possible interpretations. Let $\boldsymbol{sat}$ and $\boldsymbol{pr}$ be two vectors of thresholds of dimension k, and $0 \leq pr \leq 1$ be a single threshold. The first interpretation (namely, the *conjunctive interpretation*) requires the satisfaction semantics in each dimension $1 \leq i \leq n$ with thresholds $\boldsymbol{sat}_i$ and $\boldsymbol{pr}_i$, respectively (where $\boldsymbol{v}_i$ is the i-th component of vector $\boldsymbol{v}$). The sets of satisfying runs for each reward may even be disjoint here. The second interpretation (namely, the *joint interpretation*) requires the satisfaction semantics for all rewards at once. Precisely, it requires that, with probability at least pr, the mean-payoff value vector be at least $\boldsymbol{sat}$. The distinction of the two views (expectation vs. satisfaction) and their applicability in analysis of problems related to stochastic reactive systems has been discussed in details in [BBC+14]. While the joint interpretation of satisfaction has already been introduced and studied in [BBC+14], here we consider also the conjunctive interpretation, which was not considered in [BBC+14]. The conjunctive interpretation was considered in [FKR95], however, only a partial solution was provided, and it was mentioned that a complete solution would be very useful.

Our problem. In this work we consider a new problem that unifies the two different semantics. Intuitively, the problem we consider asks to *optimize* the expectation while *ensuring* the satisfaction. Formally, consider an MDP with n reward functions, a probability threshold vector $\boldsymbol{pr}$ (or threshold pr for joint interpretation), and a mean-payoff value threshold vector $\boldsymbol{sat}$. We consider the set of *satisfaction* strategies that ensure the satisfaction semantics. Then the optimization of the expectation is considered with respect to the satisfaction strategies. Note that if $\boldsymbol{pr}$ is $\mathbf{0}$, then the satisfaction strategies is the set of all strategies and we obtain the traditional expectation semantics as a special case.

We also consider important special cases of our problem, depending on whether there is a single reward (mono-reward) or multiple rewards (multi-reward), and whether the probability threshold is $\boldsymbol{pr} = \mathbf{1}$ (qualitative criteria) or the general case (quantitative criteria). Specifically, we consider four cases:

(1) *Mono-qual:* a single reward function and qualitative satisfaction semantics;
(2) *Mono-quant:* a single reward function and quantitative satisfaction semantics;
(3) *Multi-qual:* multiple reward functions and qualitative satisfaction semantics;
(4) *Multi-quant:* multiple reward functions and quantitative satisfaction semantics.

Note that for multi-qual and mono cases, the two interpretations (conjunctive and joint) of the satisfaction semantics coincide, whereas in the multi-quant problem (which is the most general problem) we consider both the conjunctive and the joint interpretations, separately (*multi-quant-conjunctive*, *multi-quant-joint*) as well as at once (*multi-quant-conjunctive-joint*).

Motivation. The motivation to study the problem we consider is twofold. Firstly, it presents a unifying approach that combines the two existing semantics for MDPs. Secondly and more importantly, it allows us to consider the problem of optimization along with *risk aversion.* A risk-averse strategy must ensure certain probabilistic guarantee on the payoff function. The notion of risk aversion is captured by the satisfaction semantics, and thus the problem we consider captures the notion of optimization under risk-averse strategies that provide probabilistic guarantee. The notion of *strong risk-aversion* where the probability is treated as an adversary is considered in [BFRR14], whereas we consider probabilistic (both qualitative and quantitative) guarantee for risk aversion. We now illustrate our problem with several examples.

Illustrative examples:

- For simple risk aversion, consider a single reward function modelling investment. Positive reward stands for profit, negative for loss. We aim at maximizing the expected long-run average while guaranteeing that it is non-negative with at least 95%. This is an instance of *mono-quant* with $pr = 0.95, sat = 0$.
- For more dimensions, consider the example [Put94, Problems 6.1, 8.17]. A vendor assigns to each customer either a low or a high rank. Further, there is a decision the vendor makes each year either to invest money into sending a catalogue to the customer or not. Depending on the rank and on receiving a catalogue, the customer spends different amounts for vendor's products and the rank can change. The aim is to maximize the expected profit provided the catalogue is almost surely sent with frequency at most f. This is an instance of *multi-qual.* Further, one can extend this example to only require that the catalogue frequency does not exceed f with 95% probability, but 5% best customers may still receive catalogues very often (instance of *multi-quant*).
- The following is again an instance of *multi-quant.* A gratis service for downloading is offered as well as a premium one. For each we model the throughput as rewards r_1, r_2. For the gratis service, expected throughput $1\mathit{Mbps}$ is guaranteed as well as 60% connections running on at least $0.8\mathit{Mbps}$. For the premium service, not only have we a higher expectation of $10\mathit{Mbps}$, but also 95% of the connections are guaranteed to run on at least $5\mathit{Mbps}$ and 80% on even $8\mathit{Mbps}$ (satisfaction constraints). In order to keep this guarantee, we may need to temporarily hire resources from a cloud, whose cost is modelled as a reward r_3. While satisfying

the guarantee, we want to maximize the expectation of $p_2 \cdot r_2 - p_3 \cdot r_3$ where p_2 is the price per *Mb* at which the premium service is sold and p_3 is the price at which additional servers can be hired. Note that since the percentages above are different, the constraints cannot be encoded using the joint interpretation, and conjunctive interpretation is necessary.

The basic computational questions. In MDPs with multiple mean-payoff objectives, different strategies may produce incomparable solutions. Thus, there is no "best" solution in general. Informally, the set of *achievable solutions* is the set of all vectors $\boldsymbol{v}$ such that there is a strategy that ensures the satisfaction semantics and that the expected mean-payoff value vector under the strategy is at least $\boldsymbol{v}$. The "trade-offs" among the goals represented by the individual mean-payoff objectives are formally captured by the *Pareto curve*, which consists of all maximal tuples (with respect to component-wise ordering) that are not strictly dominated by any achievable solution. Pareto optimality has been studied in cooperative game theory [Owe95] and in multi-criterion optimization and decision making in both economics and engineering [Kos88, YC03, SCK04].

We study the following fundamental questions related to the properties of strategies and algorithmic aspects in MDPs:

- *Algorithmic complexity:* What is the complexity of deciding whether a given vector represents an achievable solution, and if the answer is yes, then compute a witness strategy?
- *Strategy complexity:* What type of strategies is sufficient (and necessary) for achievable solutions?
- *Pareto-curve computation:* Is it possible to compute an approximation of the Pareto curve?

Our contributions. We provide comprehensive answers to the above questions. The main highlights of our contributions are:

- *Algorithmic complexity.* We present algorithms for deciding whether a given vector is an achievable solution and constructing a witness strategy. All our algorithms are polynomial in the size of the MDP. Moreover, they are polynomial even in the number of dimensions, except for *multi-quant* with conjunctive interpretation where it is exponential.
- *Strategy complexity.* It is known that for both expectation and satisfaction semantics with single reward, deterministic memoryless(*) strategies are sufficient [FV97, BBE10, BBC$^+$14]. We show this carries over in the *mono-qual* case only. In contrast, we show that for *mono-quant* both randomization and memory is necessary. For randomized strategies, they can be *stochastic-update*, where the memory is updated probabilistically, or *deterministic-update*, where the memory update is deterministic. We provide precise bounds on the memory size of stochastic-update strategies. Further, we show that for both mono-quant and multi-qual, deterministic-update strategies require memory size that is dependent on the MDP. Finally, we also show that deterministic-update strategies are sufficient even for *multi-quant*, thus extending the results of [BBC$^+$14].

(*)A strategy is memoryless if it is independent of the history, but depends only on the current state. A strategy that is not deterministic is called randomized.

- *Pareto-curve computation.* We show that in all cases with multiple rewards an ε-approximation of the Pareto curve can be achieved in time polynomial in the size of the MDP, exponential in the number of dimensions, and polynomial in $\frac{1}{\varepsilon}$, for $\varepsilon > 0$.

In summary, we unify the two existing semantics, present comprehensive results related to algorithmic and strategy complexities for the unifying semantics, and improve results for the existing semantics.

Technical contributions. In the study of MDPs (with single or multiple rewards), the solution approach is often by characterizing the solution as a set of linear constraints. Similar to the previous works [CMH06, EKVY08, FKN$^+$11, BBC$^+$14] we also obtain our results by showing that the set of achievable solutions can be represented by a set of linear constraints, and from the linear constraints witness strategies for achievable solutions can be constructed. However, previous work on the satisfaction semantics [BBC$^+$14, RRS15] reduces the problem to invoking linear-programming solution for each maximal end-component and a separate linear program to combine the partial results together. In contrast, we unify the solution approaches for expectation and satisfaction and provide one complete linear program for the whole problem. This in turn allows us to optimize the expectation *while* guaranteeing satisfaction. Further, this approach immediately yields a linear program where both conjunctive and joint interpretations are combined, and we can optimize any linear combination of expectations. Finally, we can also optimize the probabilistic guarantees while ensuring the required expectation. The technical device to obtain one linear program is to split the standard variables into several, depending on which subsets of constraints they help to achieve. This causes technical complications that have to be dealt with making use of conditional probability methods.

Related work. The study of Markov decision processes with multiple expectation objectives has been initiated in the area of applied probability theory, where it is known as *constrained MDPs* [Put94, Alt99]. The attention in the study of constrained MDPs has been mainly focused on restricted classes of MDPs, such as unichain MDPs, where all states are visited infinitely often under any strategy. Such a restriction guarantees the existence of memoryless optimal strategies. The more general problem of MDPs with multiple mean-payoff objectives was first considered in [Cha07] and a complete picture was presented in [BBC$^+$14]. The expectation and satisfaction semantics was considered in [BBC$^+$14], and our work unifies the two different semantics for MDPs. For general MDPs, [CMH06, CFW13] studied multiple discounted reward functions. MDPs with multiple ω-regular specifications were studied in [EKVY08]. It was shown that the Pareto curve can be approximated in polynomial time in the size of MDP and exponential in the number of specifications; the algorithm reduces the problem to MDPs with multiple reachability specifications, which can be solved by multi-objective linear programming [PY00]. In [FKN$^+$11], the results of [EKVY08] were extended to combine ω-regular and expected total reward objectives. The problem of conjunctive satisfaction was introduced in [FKR95]. They present solution for only stationary (memoryless) strategies, and explicitly mention that such strategies are not sufficient and a solution to the general problem would be very useful. They also mention that it is unlikely to be a simple extension of the single dimensional case. Our results not only present the general solution, but we also present results that combine both the conjunctive and joint satisfaction semantics along with the expectation semantics. The multiple percentile are currently considered for various objectives, such as mean-payoff, limsup, liminf, shortest path in [RRS15]. However, [RRS15] does not consider optimizing the expectation, whereas we consider maximizing expectation along with satisfaction semantics. The notion of risks has been considered in MDPs with discounted objectives [WL99], where the goal is to maximize (resp., minimize) the probability (risk) that the expected total discounted reward (resp., cost) is above (resp., below) a threshold. The notion of strong risk aversion, where for risk the probabilistic choices are treated instead as an adversary was considered in [BFRR14]. In [BFRR14] the problem was considered for single reward for mean-payoff and shortest path. In contrast, though inspired by [BFRR14], we consider risk aversion for multiple reward functions with probabilistic guarantee (instead of adversarial guarantee), which is natural for MDPs. Moreover, [BFRR14] generalizes mean-payoff games, for which no polynomial-time solution is known, whereas in our case, we present polynomial-time algorithms for the single reward case and in several cases of multiple rewards (see the first item of our contributions). Further, an independent work [CR15] extends [BFRR14] to multiple dimensions, and they also consider "beyond almost-sure threshold problem", which corresponds to the *multi-qual* problem, which is a special case of our solution. Finally, a very different notion of risk has been considered in [BCFK13], where the goal is to optimize the expectation while ensuring low variance. The problem has been considered only for single dimension, and no polynomial-time algorithm is known.

2. Preliminaries

2.1. **Basic definitions.** We mostly follow the basic definitions of [BBC$^+$14] with only minor deviations. We use $\mathbb{N}, \mathbb{Q}, \mathbb{R}$ to denote the sets of positive integers, rational and real numbers, respectively. For $n \in \mathbb{N}$, we denote $[n] = \{1, \ldots, n\}$. For a sequence $\omega = \ell_1 \ell_2 \cdots$ and $n \in \mathbb{N}$, we denote the n-th element by $\omega[n]$.

Given two vectors $\boldsymbol{v}, \boldsymbol{w} \in \mathbb{R}^k$, where $k \in \mathbb{N}$, we write $\boldsymbol{v} \geq \boldsymbol{w}$ iff $\boldsymbol{v}_i \geq \boldsymbol{w}_i$ for all $1 \leq i \leq k$, where $\boldsymbol{v}_i$ denotes the i-th component of vector $\boldsymbol{v}$. Further, $\mathbf{1}$ denotes $(1, \ldots, 1)$, and $\mathbb{1}$ denotes Kronecker's delta, i.e., $\mathbb{1}_x(x) = 1$ and $\mathbb{1}_x(y) = 0$ for $y \neq x$.

Finally, the set of all distributions over a countable set X is denoted by $\mathit{Dist}(X)$, and $d \in \mathit{Dist}(X)$ is Dirac if $d(x) = 1$ for some $x \in X$, i.e., $d = \mathbb{1}_x$.

Markov chains. A *Markov chain* is a tuple $M = (L, P, \mu)$ where L is a countable set of locations, $P : L \to \mathit{Dist}(L)$ is a probabilistic transition function, and $\mu \in \mathit{Dist}(L)$ is the initial probability distribution.

A *run* in M is an infinite sequence $\omega = \ell_1 \ell_2 \cdots$ of locations, a *path* in M is a finite prefix of a run. Each path w in M determines the set $\mathsf{Cone}(w)$ consisting of all runs that start with w. To M we associate the probability space $(\mathsf{Runs}, \mathcal{F}, \mathbb{P})$, where Runs is the set of all runs in M, $\mathcal{F}$ is the σ-field generated by all $\mathsf{Cone}(w)$, and $\mathbb{P}$ is the unique probability measure such that $\mathbb{P}(\mathsf{Cone}(\ell_1 \cdots \ell_k)) = \mu(\ell_1) \cdot \prod_{i=1}^{k-1} P(\ell_i)(\ell_{i+1})$.

Markov decision processes. A *Markov decision process* (MDP) is a tuple $G = (S, A, \mathit{Act}, \delta, s_0)$ where S is a finite set of states, A is a finite set of actions, $\mathit{Act} : S \to 2^A \setminus \{\emptyset\}$ assigns to each state s the set $\mathit{Act}(s)$ of actions enabled in s so that $\{\mathit{Act}(s) \mid s \in S\}$ is a partitioning of A, $\delta : A \to \mathit{Dist}(S)$ is a probabilistic transition function that given an action a gives a probability distribution over the successor states, and s_0 is the initial state. Note that we consider that every action is enabled in exactly one state.

A *run* in G is an infinite alternating sequence of states and actions $\omega = s_1 a_1 s_2 a_2 \cdots$ such that for all $i \geq 1$, we have $a_i \in \mathit{Act}(s_i)$ and $\delta(a_i)(s_{i+1}) > 0$. A *path* of length k in G is a finite prefix $w = s_1 a_1 \cdots a_{k-1} s_k$ of a run in G.

Strategies and plays. The semantics of MDPs is defined using the notion of strategies. Intuitively, a strategy in an MDP G is a "recipe" to choose actions. Usually, a strategy is formally defined as a function $\sigma : (SA)^* S \to \mathit{Dist}(A)$ that given a finite path w, representing the history of a play, gives a probability distribution over the actions enabled in the last state. In this paper, we adopt a slightly different (though equivalent—see [BBC+14, Section 6]) definition, which is more convenient for our setting. Let M be a countable set of *memory elements*. A *strategy* is a triple $\sigma = (\sigma_u, \sigma_n, \alpha)$, where $\sigma_u : A \times S \times \mathsf{M} \to \mathit{Dist}(\mathsf{M})$ and $\sigma_n : S \times \mathsf{M} \to \mathit{Dist}(A)$ are *memory update* and *next move* functions, respectively, and α is the initial distribution on memory elements. We require that, for all $(s, m) \in S \times \mathsf{M}$, the distribution $\sigma_n(s, m)$ assigns a positive value only to actions enabled at s, i.e. $\sigma_n(s, m) \in \mathit{Dist}(\mathit{Act}(s))$.

A *play* of G determined by a strategy σ is a Markov chain $G^\sigma = (S \times \mathsf{M} \times A, P, \mu)$, where

$$\mu(s, m, a) = \mathbb{1}_{s_0}(s) \cdot \alpha(m) \cdot \sigma_n(s, m)(a)$$
$$P(s, m, a)(s', m', a') = \delta(a)(s') \cdot \sigma_u(a, s', m)(m') \cdot \sigma_n(s', m')(a')\,.$$

Hence, G^σ starts in a location chosen randomly according to α and σ_n. In a current location (s, m, a), the next action to be performed is a, hence the probability of entering s' is $\delta(a)(s')$. The probability of updating the memory to m' is $\sigma_u(a, s', m)(m')$, and the probability of selecting a' as the next action is $\sigma_n(s', m')(a')$. Note that these choices are independent, and thus we obtain the product above. The induced probability measure is denoted by $\mathbb{P}^\sigma$ and when the initial state s is not clear from the context, we use $\mathbb{P}^\sigma_s$ to denote $\mathbb{P}^\sigma$ corresponding to the MDP where the initial state is set to s. "Almost surely" or "almost all runs" refers to happening with probability 1 according to this measure. The respective expected value of a random variable $f : \mathsf{Runs} \to \mathbb{R}$ is $\mathbb{E}^\sigma_s[f] = \int_{\mathsf{Runs}} f \, d\,\mathbb{P}^\sigma_s$ or $\mathbb{E}^\sigma[f] = \int_{\mathsf{Runs}} f \, d\,\mathbb{P}^\sigma$ for short. For $t \in \mathbb{N}$, random variables S_t, A_t return s, a, respectively, where (s, m, a) is the t-th location on the run.

Strategy types. In general, a strategy may use infinite memory M, and both σ_u and σ_n may randomize. The strategy is

- *deterministic-update*, if α is Dirac and the memory update function gives a Dirac distribution for every argument;
- *stochastic-update*, if it is not necessarily deterministic-update;
- *deterministic*, if it is deterministic-update and the next move function gives a Dirac distribution for every argument;
- *randomized*, if it is not necessarily deterministic.

We also classify the strategies according to the size of memory they use. The important subclasses of strategies are

- *memoryless* (or 1-*memory*) strategies, in which M is a singleton,
- n-*memory* strategies, in which M has exactly n elements,
- *finite-memory* strategies, in which M is finite, and
- *Markov* strategies, in which $\mathsf{M} = \mathbb{N}$ and $\sigma_u(\cdot, \cdot, n)(n + 1) = 1$.

Markov strategies have a nice structure: they only need a counter and to know the current state [FV97].

End components. A set $T \cup B$ with $\emptyset \neq T \subseteq S$ and $B \subseteq \bigcup_{t \in T} Act(t)$ is an *end component* of G if (1) for all $a \in B$, whenever $\delta(a)(s') > 0$ then $s' \in T$; and (2) for all $s, t \in T$ there is a path $\omega = s_1 a_1 \cdots a_{k-1} s_k$ such that $s_1 = s$, $s_k = t$, and all states and actions that appear in ω belong to T and B, respectively. An end component $T \cup B$ is a *maximal end component (MEC)* if it is maximal with respect to the subset ordering. Given an MDP, the set of MECs is denoted by MEC. Finally, if (S, A) is a MEC, we call the MDP *strongly connected.*

Remark 2.1. The maximal end component (MEC) decomposition of an MDP, i.e., the computation of MEC, can be achieved in polynomial time [CY95]. For improved algorithms for general MDPs and various special cases see [CH11, CH12, CH14, CL13].

Analogously, for a finite-memory strategy σ, a *bottom strongly connected component* (BSCC) of G^σ is a subset of locations $W \subseteq S \times \mathsf{M} \times A$ such that (i) for all $\ell_1 \in W$ and $\ell_2 \in S \times \mathsf{M} \times A$, if there is a path from ℓ_1 to ℓ_2 then $\ell_2 \in W$, and (ii) for all $\ell_1, \ell_2 \in W$ we have a path from ℓ_1 to ℓ_2. Every BSCC W determines a unique end component $\{s, a \mid (s, m, a) \in W\}$ of G, and we sometimes do not strictly distinguish between W and its associated end component.

For $C \in \mathsf{MEC}$, let

$$\Omega_C = \{\omega \in \mathsf{Runs} \mid \exists n_0 : \forall n > n_0 : \omega[n] \in C\}$$

denote the set of runs with a suffix in C. Similarly, we define Ω_D for a BSCC D. Since almost every run eventually remains in a MEC, e.g. [CY98, Proposition 3.1], $\{\Omega_C \mid C \in \mathsf{MEC}\}$ "partitions" almost all runs. More precisely, for every strategy, each run belongs to exactly one Ω_C almost surely; i.e. a run never belongs to two Ω_C's and for every σ, we have $\mathbb{P}^\sigma\left[\bigcup_{C \in \mathsf{MEC}} \Omega_C\right] = 1$. Therefore, actions that are not in any MEC are almost surely taken only finitely many times.

2.2. **Problem statement.**

In order to define our problem, we first briefly recall how long-run average can be defined. Let $G = (S, A, Act, \delta, s_0)$ be an MDP, $n \in \mathbb{N}$ and $\boldsymbol{r} : A \to \mathbb{Q}^n$ an n-dimensional *reward function.* Since the random variable given by the limit-average function $\mathrm{lr}(\boldsymbol{r}) = \lim_{T \to \infty} \frac{1}{T} \sum_{t=1}^{T} \boldsymbol{r}(A_t)$ may be undefined for some runs, we consider maximizing the respective point-wise limit inferior:

$$\mathrm{lr}_{\inf}(\boldsymbol{r}) = \liminf_{T \to \infty} \frac{1}{T} \sum_{t=1}^{T} \boldsymbol{r}(A_t)$$

i.e. for each $i \in [n]$ and $\omega \in \mathsf{Runs}$, we have $\mathrm{lr}_{\inf}(\boldsymbol{r})(\omega)_i = \liminf_{T \to \infty} \frac{1}{T} \sum_{t=1}^{T} \boldsymbol{r}(A_t(\omega))_i$. Similarly, we could define $\mathrm{lr}_{\sup}(\boldsymbol{r}) = \limsup_{T \to \infty} \frac{1}{T} \sum_{t=1}^{T} \boldsymbol{r}(A_t)$. However, maximizing limit superior is less interesting, see [BBC$^+$14]. Further, the respective minimizing problems can be solved by maximization with opposite rewards.

This paper is concerned with the following tasks:

Realizability (multi-quant-conjunctive): Given an MDP, $n \in \mathbb{N}, \boldsymbol{r} : A \to \mathbb{Q}^n$, $\boldsymbol{exp} \in \mathbb{Q}^n, \boldsymbol{sat} \in \mathbb{Q}^n, \boldsymbol{pr} \in ([0,1] \cap \mathbb{Q})^n$, decide whether there is a strategy σ such that $\forall i \in [n]$

- $\mathbb{E}^\sigma[\mathrm{lr}_{\inf}(\boldsymbol{r})_i] \geq \boldsymbol{exp}_i\,,$ (EXP)
- $\mathbb{P}^\sigma[\mathrm{lr}_{\inf}(\boldsymbol{r})_i \geq \boldsymbol{sat}_i] \geq \boldsymbol{pr}_i\,.$ (conjunctive-SAT)

Witness strategy synthesis: If realizable, construct a strategy satisfying the requirements.

ε-witness strategy synthesis: If realizable, construct a strategy satisfying the requirements with $\boldsymbol{exp} - \varepsilon \cdot \mathbf{1}$ and $\boldsymbol{sat} - \varepsilon \cdot \mathbf{1}$.

We are mostly interested in **(multi-quant-conjunctive)** as it is the core of all other discussed problems. However, we also consider the following important special cases:

(multi-qual) :	$\boldsymbol{pr} = \mathbf{1}\,,$
(mono-quant):	$n = 1\,,$
(mono-qual) :	$n = 1, \boldsymbol{pr} = 1\,.$

Additionaly, we are also interested in variants of **(multi-quant-conjunctive)**. Firstly, in **(multi-quant-joint)**, the constraint (conjunctive-SAT) is *replaced* by

$$\mathbb{P}^\sigma[\mathrm{lr}_{\inf}(\boldsymbol{r}) \geq \boldsymbol{sat}] \geq pr \qquad \text{(joint-SAT)}$$

for $pr \in [0,1]$. Secondly, **(multi-quant-conjunctive-joint)** arises by *adding* (joint-SAT) constraint $\mathbb{P}^\sigma\Big[\mathrm{lr}_{\inf}(\boldsymbol{r}) \geq \widetilde{\boldsymbol{sat}}\Big] \geq \widetilde{pr}$ for $\widetilde{pr} \in [0,1] \cap \mathbb{Q}$ and $\widetilde{\boldsymbol{sat}} \in \mathbb{Q}^n$. The relationship between the problems is depicted in Fig. 1.

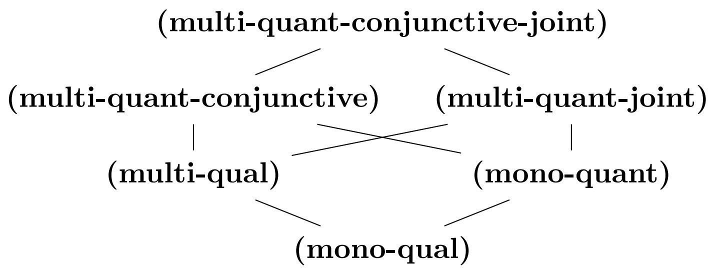
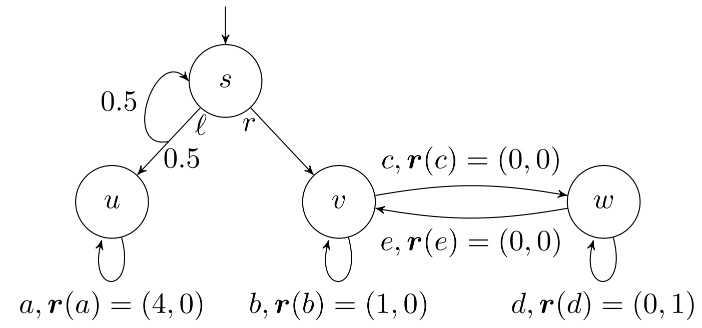


FIGURE 1. Relationship of the defined problems with lower problems being specializations of the higher ones

Furthermore, each of the three constraints (EXP), (conjunctive-SAT), and (joint-SAT) defines the respective decision problem given solely by that constraint. Each of these three problems is a special case of **(multi-quant-conjunctive-joint)** where the other constraints are trivial (e.g. requiring the average reward be greater or equal to the minimum reward of the MDP). Finally, apart from decision problems, one often considers optimization problems, where the task is to maximize the parameters so that the answer to the decision problem is still positive. Observe that since optimization in multi-dimensional setting cannot in general produce a single "best" solution, one can consider Pareto curves, which are sets of all component-wise optimal and mutually incomparable solutions to the optimization problem.

Example 2.2 (Running example)**.** We illustrate **(multi-quant-conjunctive)** with an MDP of Fig. 2 with $n = 2$, rewards as depicted, and $\boldsymbol{exp} = (1.1, 0.5), \boldsymbol{sat} = (0.5, 0.5), \boldsymbol{pr} = (0.8, 0.8)$. Observe that rewards of actions ℓ and r are irrelevant as these actions can almost surely be taken only finitely many times.



s
0.5
ℓ
r
0.5
u
v
w
$c, \boldsymbol{r}(c) = (0, 0)$
$e, \boldsymbol{r}(e) = (0, 0)$
$a, \boldsymbol{r}(a) = (4, 0)$
$b, \boldsymbol{r}(b) = (1, 0)$
$d, \boldsymbol{r}(d) = (0, 1)$

Figure 2. An MDP with two-dimensional rewards

This instance is realizable and the witness strategy has the following properties. The strategy plays three "kinds" of runs. Firstly, due to $\boldsymbol{pr} = (0.8, 0.8)$, with probability at least $0.8 + 0.8 - 1 = 0.6$ runs have to jointly surpass both satisfaction thresholds (at the same time), i.e. exceed the vector $(0.5, 0.5)$. This is only possible in the right MEC by playing each b and d half of the time and switching between them with a decreasing frequency, so that the frequency of c, e is in the limit 0. Secondly, in order to ensure the expectation of the first reward, we reach the left MEC with probability 0.2 and play a. Thirdly, with probability 0.2 we reach again the right MEC but only play d with frequency 1, ensuring the expectation of the second reward.

In order to play these three kinds of runs, in the first step in s we take ℓ with probability 0.4 (arriving to u with probability 0.2) and r with probability 0.6, and if we return back to s we play r with probability 1. If we reach the MEC on the right, we toss a biased coin and with probability 0.25 we go to w and play the third kind of runs, and with probability 0.75 play the first kind of runs.

Observe that although both the expectation and satisfaction value thresholds for the second reward are 0.5, the only solution is not to play all runs with this reward, but some with a lower one and some with a higher one. Also note that each of the three types of runs must be present in any witness strategy. Most importantly, in the MEC at state w we have to play in two different ways, depending on which subset of value thresholds we intend to satisfy on each run. Also note that in order to do that, we use memory with stochastic update. △

3. Solution

In this section, we briefly recall a solution to a previously considered problem and show our solution to the more general **(multi-quant-conjunctive)** realizability problem, along with an overview of the correctness proof. The solution to the other variants is derived and a detailed analysis of the special cases and the respective complexities is given in Section 6.

3.1. **Previous results.**

3.1.1. *Linear programming for expectation semantics.* In [BBC$^+$14], a solution to the (EXP) constraint has been given. The existence of a witness strategy was shown equivalent to the existence of a solution to the linear program in Fig. 3.

Requiring all variables y_a, y_s, x_a for $a \in A, s \in S$ be non-negative, the program is the following:

(1) transient flow: for $s \in S$

$$\mathbb{1}_{s_0}(s) + \sum_{a \in A} y_a \cdot \delta(a)(s) = \sum_{a \in Act(s)} y_a + y_s$$

(2) almost-sure switching to recurrent behaviour:

$$\sum_{s \in C \in \mathsf{MEC}} y_s = 1$$

(3) probability of switching in a MEC is the frequency of using its actions: for $C \in \mathsf{MEC}$

$$\sum_{s \in C} y_s = \sum_{a \in C} x_a$$

(4) recurrent flow: for $s \in S$

$$\sum_{a \in A} x_a \cdot \delta(a)(s) = \sum_{a \in Act(s)} x_a$$

(5) expected rewards:

$$\sum_{a \in A} x_a \cdot \boldsymbol{r} \geq \boldsymbol{exp}$$

FIGURE 3. Linear program of [BBC$^+$14] for (EXP)

Intuitively, x_a is the expected frequency of using a on the long run; Equation 4 thus expresses the recurrent flow in MECs and Equation 5 the expected long-run average reward. However, before we can play according to x-variables, we have to reach MECs and switch from the transient behaviour to this recurrent behaviour. Equation 1 expresses the transient flow before switching. Variables y_a are the expected number of using a until we switch to the recurrent behaviour in MECs and y_s is the probability of this switch upon reaching s. To relate y- and x-variables, Equation 3 states that the probability to switch within a given MEC is the same whether viewed from the transient or recurrent flow perspective. Actually, one could eliminate variables y_s and use directly x_a in Equation 1 and leave out Equation 3 completely, in the spirit of [Put94]. However, the form with explicit y_s is more convenient for correctness proofs. Finally, Equation 2 states that switching happens almost surely. Note that summing Equation 1 over all $s \in S$ yields $\sum_{s \in S} y_s = 1$. Since y_s can be shown to equal 0 for state s not in MEC, Equation 2 is redundant, but again more convenient.

The solution above builds on the work [EKVY08], which studied MDPs with multiple reachability and ω-regular specifications. It has inspired Equation 1 as well as computation of the Pareto curve. It was shown that the Pareto curve can be approximated in polynomial time in the size of MDP and exponential in the number of specifications; the algorithm reduces the problem to MDPs with multiple reachability specifications, which can be solved by multi-objective linear programming [PY00].

3.1.2. *Linear programming for satisfaction semantics.* Apart from considering (EXP) separately, [BBC$^+$14] also considers the constraint (joint-SAT) separately. While the former was solved using the linear program above, the latter required a reduction to one linear program per each MEC and another one to combine the results. More precisely, for each MEC we first decide whether there is a strategy exceeding the threshold. Second, we maximize the probability to reach these MECs. Similarly, in [RRS15], for each MEC we decide for every subset of thresholds whether there is a strategy exceeding them. The results are again combined in a linear program for reachability.

In contrast, we shall provide a single linear program for the **(multi-quant-conjunctive)** problem, unifying the solution approaches for expectation and satisfaction problem. This in turn allows us to optimize the expectation *while* guaranteeing satisfaction. Further, this approach immediately yields a linear program where both conjunctive and joint interpretations are combined, and we can optimize any linear combination of expectations. Finally, we can also optimize the probabilistic guarantees while ensuring the required expectation. For greater detail, see Section 3.4.

3.2. **Our unifying solution.** There are two main tricks to incorporate the satisfaction semantics. The first one is to ensure that a flow exceeds the value threshold. We first explain it on the qualitative case.

3.2.1. *Solution to* ***(multi-qual)****.* When the additional constraint (SAT) is added so that almost all runs satisfy $\mathrm{lr}_{\mathrm{inf}}(\boldsymbol{r}) \geq \boldsymbol{sat}$, then the linear program of Fig. 3 shall be extended with the following additional equation:

6. almost-sure satisfaction: for $C \in \mathsf{MEC}$

$$\sum_{a\in C} x_a \cdot \boldsymbol{r}(a) \geq \sum_{a \in C} x_a \cdot \boldsymbol{sat}$$

Note that x_a represents the absolute frequency of playing a (not relative within the MEC). Intuitively, Equation 6 thus requires in each MEC the average reward be at least $\boldsymbol{sat}$. Here we rely on the non-trivial fact, that in a MEC, actions can be played on almost all runs with the given frequencies for any flow, see Corollary 5.5.

The second trick ensures that each conjunct in the satisfaction constraint can be handled separately and, consequently, that the probability threshold can be checked.

3.2.2. *Solution to* ***(multi-quant-conjunctive)****.* When each value threshold $\boldsymbol{sat}_i$ comes with a non-trivial probability threshold $\boldsymbol{pr}_i$, some runs may and some may not have the long-run average reward exceeding $\boldsymbol{sat}_i$. In order to speak about each group, we split the set of runs, for each reward, into parts which do and which do not exceed the threshold.

Technically, we keep Equations 1–5 as well as 6, but split x_a into $x_{a,N}$ for $N \subseteq [n]$, where N describes the subset of exceeded thresholds; similarly for y_s. The linear program L then takes the form displayed in Fig. 4.

Intuitively, only the runs in the appropriate "N-classes" are required in Equation 6 to have long-run average rewards exceeding the satisfaction value threshold. However, only

Requiring all variables $y_a, y_{s,N}, x_{a,N}$ for $a \in A, s \in S, N \subseteq [n]$ be non-negative, the program is the following:

(1) transient flow: for $s \in S$

$$\mathbb{1}_{s_0}(s) + \sum_{a \in A} y_a \cdot \delta(a)(s) = \sum_{a \in Act(s)} y_a + \sum_{N \subseteq [n]} y_{s,N}$$

(2) almost-sure switching to recurrent behaviour:

$$\sum_{\substack{s \in C \in \mathsf{MEC} \\ N \subseteq [n]}} y_{s,N} = 1$$

(3) probability of switching in a MEC is the frequency of using its actions: for $C \in \mathsf{MEC}, N \subseteq [n]$

$$\sum_{s \in C} y_{s,N} = \sum_{a \in C} x_{a,N}$$

(4) recurrent flow: for $s \in S, N \subseteq [n]$

$$\sum_{a \in A} x_{a,N} \cdot \delta(a)(s) = \sum_{a \in Act(s)} x_{a,N}$$

(5) expected rewards:

$$\sum_{\substack{a \in A, \\ N \subseteq [n]}} x_{a,N} \cdot \boldsymbol{r}(a) \geq \boldsymbol{exp}$$

(6) commitment to satisfaction: for $C \in \mathsf{MEC}$, $N \subseteq [n]$, $i \in N$

$$\sum_{a \in C} x_{a,N} \cdot \boldsymbol{r}(a)_i \geq \sum_{a \in C} x_{a,N} \cdot \boldsymbol{sat}_i$$

(7) satisfaction: for $i \in [n]$

$$\sum_{\substack{a \in A, \\ N \subseteq [n] : i \in N}} x_{a,N} \geq \boldsymbol{pr}_i$$

FIGURE 4. Linear program L for **(multi-quant-conjunctive)**

the appropriate "N-classes" are considered for surpassing the probabilistic threshold in Equation 7.

Theorem 3.1. Given a **(multi-quant-conjunctive)** realizability problem, the respective system L (in Fig. 4) satisfies the following:

(1) The system L is constructible and solvable in time polynomial in the size of G and exponential in n.
(2) Every witness strategy induces a solution to L.
(3) Every solution to L effectively induces a witness strategy.

Example 3.2 (Running example)**.** The linear program L for Example 2.2 is shown in Appendix A. Here we spell out some useful points we need later: Equation 1 for state s

$$1 + 0.5 y_\ell = y_\ell + y_r + y_{s,\emptyset} + y_{s,\{1\}} + y_{s,\{2\}} + y_{s,\{1,2\}}$$

expresses the Kirchhoff's law for the flow through the initial state. Equation 6 for the MEC $C = \{v, w, b, c, d, e\}$, $N = \{1, 2\}$, $i = 1$

$$x_{b,\{1,2\}} \cdot 1 \geq (x_{b,\{1,2\}} + x_{c,\{1,2\}} + x_{d,\{1,2\}} + x_{e,\{1,2\}}) \cdot 0.5$$

expresses that runs ending up in C and satisfying both satisfaction value thresholds have to use action b at least half of the time. The same holds for d and thus actions c, e must be played with zero frequency on these runs. Equation 7 for $i = 1$ sums up the gain of all actions on runs that have committed to exceed the satisfaction value threshold either for the first reward, or for the first *and* the second reward.

Moreover, we show later in Lemma 5.1, that variables $x_{\ell,N}, x_{r,N}$ for any $N \subseteq [n]$ can be omitted from the system as they are zero for any solution. Intuitively, transient actions cannot be used in the recurrent flows. △

3.3. **Proof overview.** Here, we briefly describe the main ideas of the proof of Theorem 3.1.

The first point. The complexity follows immediately from the syntax of L and the existence of a polynomial-time algorithm for linear programming [Sch86].

The second point. Given a witness strategy σ, we construct values for variables so that a valid solution is obtained. The technical details can be found in Section 4.

The proof of [BBC$^+$14, Proposition 4.5], which inspires our proof, sets the values of x_a to be the expected frequency of using a by σ, i.e.

$$\lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} \mathbb{P}^\sigma[A_t = a]$$

Since this Cesaro limit (expected frequency) may not be defined, a suitable value $f(a)$ between the limit inferior and superior has to be taken. In contrast to the approach of [BBC$^+$14], we need to distinguish among runs exceeding various subsets of the value thresholds $\boldsymbol{sat}_i, i \in [n]$. For $N \subseteq [n]$, we call a run N-*good* if $\mathrm{lr}_{\inf}(\boldsymbol{r})_i \geq \boldsymbol{sat}_i$ for exactly all $i \in N$. N-good runs thus *jointly* satisfy the N-subset of the constraints. Now instead of using frequencies $f(a)$ of each action a, we use frequencies $f_N(a)$ of the action a on N-good runs separately, for each N. This requires some careful conditional probability considerations, in particular for Equations 1, 4, 6 and 7.

Example 3.3 (Running example)**.** The strategy of Example 2.2 induces the following x-values. For instance, action a is played with a frequency 1 on runs of measure 0.2, hence $x_{a,\{1\}} = 0.2$ and $x_{a,\emptyset} = x_{a,\{2\}} = x_{a,\{1,2\}} = 0$. Action d is played with frequency 0.5 on runs of measure 0.6 exceeding both value thresholds, and with frequency 1 on runs of measure 0.2 exceeding only the second value threshold. Consequently, $x_{d,\{1,2\}} = 0.5 \cdot 0.6 = 0.3$ and $x_{d,\{2\}} = 0.2$ whereas $x_{d,\emptyset} = x_{d,\{1\}} = 0$. △

Values for y-variables are derived from the expected number of taking actions during the "transient" behaviour of the strategy. Since the expectation may be infinite in general, an equivalent strategy is constructed, which is memoryless in the transient part, but switches to the recurrent behaviour in the same way. Then the expectations are finite and the result of [EKVY08] yields values satisfying the transient flow equation. Further, similarly as for x-values, instead of simply switching to recurrent behaviour in a particular MEC, we consider switching in a MEC *and* the set N for which the following recurrent behaviour is N-good.

Example 3.4 (Running example)**.** The strategy of Example 2.2 plays in s for the first time ℓ with probability 0.4 and r with 0.6, and next time r with probability 1. This is equivalent to a memoryless strategy playing ℓ with 1/3 and r with 2/3. Indeed, both ensure reaching the left MEC with 0.2 and the right one with 0.8. Consequently, for instance for r, the expected number of taking this action is

$$y_r = \frac{2}{3} + \frac{1}{6} \cdot \frac{2}{3} + \left(\frac{1}{6}\right)^2 \cdot \frac{2}{3} + \cdots = \frac{5}{6}\,.$$

The values $y_{u,\{1\}} = 0.2$, $y_{v,\{1,2\}} = 0.6$, $y_{v,\{2\}} = 0.2$ are given by the probability measures of each "kind" of runs (see Example 2.2). △

The third point. Given a solution to L, we construct a witness strategy σ, which has a particular structure. The technical details can be found in Section 5. The general pattern follows the proof method of [BBC$^+$14, Proposition 4.5], but there are several important differences.

First, a strategy is designed to behave in a MEC so that the frequencies of actions match the x-values. The structure of the proof differs here and we focus on underpinning the following key principle. Note that the flow described by x-variables has in general several disconnected components within the MEC, and thus actions connecting them must not be played with positive frequency. Yet there are strategies that on almost all runs play actions of all components with exactly the given frequencies. The trick is to play the "connecting" actions with an increasingly negligible frequency. As a result, the strategy visits all the states of the MEC infinitely often, as opposed to strategies generated from the linear program in Fig. 3 in [BBC$^+$14], which is convenient for the analysis.

Second, the construction of the recurrent part of the strategy as well as switching to it has to reflect again the different parts of L for different N, resulting in N-good behaviours.

Example 3.5 (Running example)**.** A solution with $x_{b,\{1,2\}} = 0.3, x_{d,\{1,2\}} = 0.3$ induces two disconnected flows. Each is an isolated loop, yet we can play a strategy that plays both actions exactly half of the time. We achieve this by playing actions c, e with probability $1/2^k$ in the $k-$th step. In Section 5 we discuss the construction of the strategy from the solution in greater detail, necessary for later complexity discussion. △

3.4. **Important aspects of our approach and its consequences.**

We now explain some important conceptual aspects of our result. The previous proof idea from [BBC$^+$14] is as follows: (1) The problem for expectation semantics is solved by a linear program. (2) The problem for satisfaction semantics is solved as follows: each MEC is considered, solved separately using a linear program, and then a reachability problem is solved using a different linear program. In comparison, our proof has two conceptual steps. Since our goal is to optimize the expectation (which intuitively requires a linear program), the first step is to come up with a single linear program for satisfaction semantics. The second step is to come up with a linear program that unifies the linear program for expectation semantics and the linear program for satisfaction semantics, allowing us to maximize expectation while ensuring satisfaction.

Since our solution captures all the frequencies separately within one linear program, we can work with all the flows at once. This has several consequences:

- While all the hard constraints are given as a part of the problem, we can easily find maximal solution with respect to a weighted reward expectation, i.e. $\boldsymbol{w} \cdot \mathrm{lr}_{\inf}(\boldsymbol{r})$, where $\boldsymbol{w}$ is the vector of weights for each reward dimension. Indeed, it can be expressed as the objective function $\boldsymbol{w} \cdot \sum_{a,N} x_{a,N} \cdot \boldsymbol{r}(a)$ of the linear program. Further, it is also relevant for the construction of the Pareto curve.
- We can also optimize satisfaction guarantees for given expectation thresholds. For more detail, see Section 8.
- We can easily add more satisfaction constraints (with different thresholds) on the same resource as well as add joint constraints of the form $\mathbb{P}^\sigma\left[\bigwedge_{k_i} \mathrm{lr}_{\inf}(\boldsymbol{r}_{k_i}) \geq pr\right]$. Both can be solved by adding a copy of Equation 7 for each subset N of all the constraints.
- The number of variables used in the linear program immediately yields an upper bound on the computational complexity of various subclasses of the general problem. Several polynomial bounds are proven in Section 6. △

4. Proof of Theorem 3.1: Witness strategy induces solution to L

Now we present the technical proof of Theorem 3.1. We start with the second point and show how to construct a solution to L from a witness strategy.

Let σ be a strategy such that $\forall i \in [n]$

- $\mathbb{P}^\sigma[\mathrm{lr}_{\inf}(\boldsymbol{r})_i \geq \boldsymbol{sat}_i] \geq \boldsymbol{pr}_i$
- $\mathbb{E}^\sigma[\mathrm{lr}_{\inf}(\boldsymbol{r})_i] \geq \boldsymbol{exp}_i$

We construct a solution to the system L. The proof method roughly follows that of [BBC⁺14, Proposition 4.5]. However, separate flows for "N-good" runs require some careful conditional probability considerations, in particular for Equations 4, 6 and 7.

4.1. Recurrent behaviour and Equations 4–7.

We start with constructing values for variables $x_{a,N}, a \in A, N \subseteq [n]$.

In general, the frequencies of the actions may not be well defined, because the defining limits may not exist. Further, it may be unavoidable to have different frequencies for several sets of runs of positive measure. There are two tricks to overcome this difficulty. Firstly, we partition the runs into several classes depending on which parts of the objective they achieve. Secondly, within each class we pick suitable values lying between $\mathrm{lr}_{\inf}(\boldsymbol{r})$ and $\mathrm{lr}_{\sup}(\boldsymbol{r})$ of these runs. In order to achieve the first point, we define for $N \subseteq [n]$,

$$\Omega_N = \{\omega \in \mathsf{Runs} \mid \forall i \in N : \mathrm{lr}_{\inf}(\boldsymbol{r})(\omega)_i \geq \boldsymbol{sat}_i \wedge \forall i \notin N : \mathrm{lr}_{\inf}(\boldsymbol{r})(\omega)_i < \boldsymbol{sat}_i\}$$

Then Ω_N, $N \subseteq [n]$ form a partitioning of Runs. Further, observe that runs of Ω_N are the runs where joint satisfaction holds, for all rewards $i \in N$. This is important for the algorithm for **(multi-quant-joint)** from Section 6.

In order to achieve the second point, we define $f_N(a)$, for every a, to be lying between values $\liminf_{T\to\infty} \frac{1}{T}\sum_{t=1}^{T} \mathbb{P}^\sigma[A_t = a \cap \Omega_N]$ and $\limsup_{T\to\infty} \frac{1}{T}\sum_{t=1}^{T} \mathbb{P}^\sigma[A_t = a \cap \Omega_N]$, which can be safely substituted for $x_{a,N}$ in L. Let A be written as $\{a_1, a_2, \ldots, a_{|A|}\}$ and let us first consider the case when $\mathbb{P}^\sigma[\Omega_N] > 0$. Since every bounded infinite sequence contains an infinite convergent subsequence, there is an increasing sequence of indices, $T_0^1, T_1^1, T_2^1 \ldots,$

such that $\lim_{\ell\to\infty}\frac{1}{T^1_\ell}\sum_{t=1}^{T^1_\ell}\mathbb{P}^\sigma[A_t=a_1\mid\Omega_N]$ is well defined. Then we can choose a subsequence $T^2_0,T^2_1,T^2_2\ldots$ of the sequence $T^1_0,T^1_1,T^1_2\ldots$ so that $\lim_{\ell\to\infty}\frac{1}{T^1_\ell}\sum_{t=1}^{T^1_\ell}\mathbb{P}^\sigma[A_t=a_1\mid\Omega_N]$ is well defined, too. We continue this process for all actions and finally define the sequence $T_0,T_1,T_2\ldots$ to be $T^{|A|}_0,T^{|A|}_1,T^{|A|}_2\ldots$. Consequently, for each action $a\in A$, the following limit exists

$$f_N(a):=\lim_{\ell\to\infty}\frac{1}{T_\ell}\sum_{t=1}^{T_\ell}\mathbb{P}^\sigma[A_t=a\mid\Omega_N]\cdot\mathbb{P}^\sigma[\Omega_N]$$

and we set for all $a\in A$

$$x_{a,N}:=f_N(a)$$

Finally, for N such that $\mathbb{P}^\sigma[\Omega_N]=0$, we set $x_{a,N}:=0$. Note that since actions not in MECs are almost surely taken only finitely many times, we have

$$x_{a,N}=0\qquad\text{for } a\notin\bigcup\mathsf{MEC}, N\subseteq[n]\tag{4.1}$$

We show that (in)equations 4–7 of L are satisfied.

Equation 4. For $N\subseteq[n],t\in\mathbb{N},a\in A,s\in S$, let

$$\Delta^N_t(a)(s):=\mathbb{P}^\sigma[S_{t+1}=s\mid A_t=a,\ \Omega_N]$$

denote the "transition probability" at time t restricted to runs in Ω_N. In general, $\Delta^N_i(a)(s)$ may be different from $\delta(a)(s)$. However, we show that if we use the action a with positive frequency then $\Delta^N_i(a)(s)$ approximates $\delta(a)(s)$.

Example 4.1. Consider an action a with $\delta(a)(u)=0.5$. Therefore, we have $\mathbb{P}^\sigma[S_2=u\mid A_1=a]=0.5$. It may well be that for some set $\Omega\subseteq\mathsf{Runs}$ we have $\mathbb{P}^\sigma[S_2=u\mid A_1=a,\ \Omega]=1$, but then $\mathbb{P}^\sigma[\Omega]\leq 0.5$. Similarly, if $\mathbb{P}^\sigma[S_2=u\mid A_1=a,\ \Omega]=\mathbb{P}^\sigma[S_3=u\mid A_2=a,\ \Omega]=1$ then $\mathbb{P}^\sigma[\Omega]\leq 0.25$, and so on. In general, whenever $\mathbb{P}^\sigma[\Omega]>0$, the transition probabilities on Ω cannot differ from the actual transition probabilities too much all the time. △

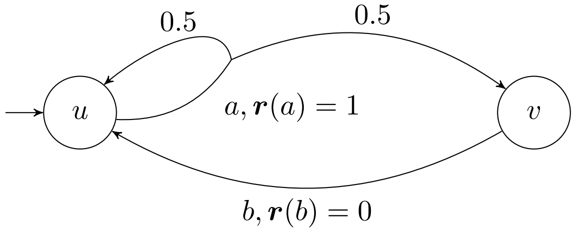


FIGURE 5. An MDP illustrating Δ

We first consider a simpler problem:

Lemma 4.2. Let $(\Delta_t)_{t\in\mathbb{N}}$ be i.i.d. Bernoulli variables with expectation $\delta=\mathbb{E}[\Delta_t]$. Then for any event Ω with $\mathbb{P}[\Omega]>0$, we have $\lim_{t\to\infty}\mathbb{E}_\Omega[\Delta_t]=\delta$.

Proof. For a contradiction, let w.l.o.g. $\limsup_{t\to\infty} \mathbb{E}_\Omega[\Delta_t] = \delta+3\varepsilon$. (If $\limsup_{t\to\infty} \mathbb{E}_\Omega[\Delta_t] < \delta$, we can consider the variables $1-\Delta_t$ with this property). Moreover, we may safely assume that $\mathbb{E}_\Omega[\Delta_t] \geq \delta + 2\varepsilon$ for all $t \in \mathbb{N}$, otherwise we consider the respective subsequence. Let $High_i \subseteq \Omega$ be the set of runs of Ω such that $\frac{1}{i}\sum_{t=1}^{i} \Delta_t > \delta + \varepsilon$ and similarly $Normal_i \subseteq \Omega$ be the set of runs of Ω such that $\frac{1}{i}\sum_{t=1}^{i} \Delta_t \leq \delta+\varepsilon$. Clearly, $\Omega = High_i \uplus Normal_i$ for every i. Then

$$\begin{aligned}\delta + 2\varepsilon &\leq \frac{1}{i}\sum_{t=1}^{i} \mathbb{E}_\Omega[\Delta_t] = \frac{1}{i}\mathbb{E}_\Omega\left[\sum_{t=1}^{i} \Delta_t\right] \\ &= \frac{\frac{1}{i}\mathbb{E}_{High_i}[\sum_{t=1}^{i}\Delta_t]\cdot\mathbb{P}[High_i] + \frac{1}{i}\mathbb{E}_{Normal_i}[\sum_{t=1}^{i}\Delta_t]\cdot\mathbb{P}[Normal_i]}{\mathbb{P}[High_i]+\mathbb{P}[Normal_i]} \\ &\leq \frac{1\cdot\mathbb{P}[High_i] + (\delta+\varepsilon)\cdot\mathbb{P}[Normal_i]}{\mathbb{P}[High_i]+\mathbb{P}[Normal_i]}\end{aligned}$$

Altogether, by comparing the first and the last expression, we get

$$\mathbb{P}[Normal_i] \leq \frac{1-\delta-2\varepsilon}{\varepsilon}\cdot\mathbb{P}[High_i] \tag{4.2}$$

where the fraction is constant for all i. Since by the law of large numbers $\lim_{i\to\infty}\mathbb{P}[High_i] = 0$, we obtain $\lim_{i\to\infty}\mathbb{P}[Normal_i] = 0$ and thus $\mathbb{P}[\Omega] = 0$, a contradiction. □

Now we apply the preceding lemma to MDPs:

Lemma 4.3. Let $N \subseteq [n]$ be such that $\mathbb{P}^\sigma[\Omega_N] > 0$. Then for every $a \in A, s \in S$, we have $\lim_{t\to\infty} \mathbb{P}^\sigma[A_t = a \mid \Omega_N]\cdot|\Delta_t^N(a)(s) - \delta(a)(s)| = 0$.

Proof plan. Note that if $\mathbb{P}^\sigma[A_t = a \mid \Omega_N] = 1$ for all t then the result follows directly from the previous lemma where we set $\Delta_t(\omega)$ to 1 if $S_{t+1} = s$ and 0 otherwise. Indeed, then $\mathbb{E}[\Delta_t] = \delta(a)(s)$ and $\mathbb{E}_{\Omega_N}[\Delta_t] = \Delta_t^N(a)(s)$. Consequently, $\lim_{t\to\infty}\mathbb{P}^\sigma[A_t = a \mid \Omega_N]\cdot|\Delta_t^N(a)(s) - \delta(a)(s)| = 1\cdot 0$.

In the general case, the probability of taking a on the runs can vary over time. In order to cope with that, we consider sets $I \subset \mathbb{N}$ of positions where a is taken with high enough probability (i.e., in "many" runs). The first step of the proof is thus to derive (4.3), an analogue of (4.2), but now relativized to positions in I. In the previous lemma, the second step consisted in applying the law of large numbers to conclude that probability of overly high preference of some outcome has zero probability, causing a contradiction with (4.2). In this proof, the second step will require more math to conclude that, due to the relativization.

Proof. Suppose for a contradiction, that for some $a \in A, s \in S$ there are infinitely many t for which $\mathbb{P}^\sigma[A_t = a \mid \Omega_N]\cdot|\Delta_t^N(a)(s) - \delta(a)(s)| > \xi$ for some $\xi > 0$. Denote the set of these t's by T. Since both factors are bounded by 0 and 1, there are $\zeta > 0$ and $\varepsilon > 0$ such that for all $t \in T$ we have $\mathbb{P}^\sigma[A_t = a \mid \Omega_N] > \zeta$ and w.l.o.g. $\Delta_t^N(a)(s) > \delta(a)(s) + 2\varepsilon$ (if $\Delta_t^N(a)(s) < \delta(a)(s)$ then there is another successor s' of a with this property). Consequently, for every $t \in T$, we have

$$\frac{\mathbb{P}^\sigma[\Omega_N \cap A_t = a \cap S_{t+1} = s]}{\mathbb{P}^\sigma[\Omega_N \cap A_t = a]} > \delta(a)(s) + 2\varepsilon$$

First step. Now we derive (4.3), a version of (4.2) relativized to finite sets $I \subseteq T$. The positive probability of taking a in these positions guarantees that overly high preference of the outcome s is well defined.

Formally, similarly to the previous inequality for each $t \in T$, the same holds for the average over any finite set of indices $I \subseteq T$:

$$\delta(a)(s) + 2\varepsilon < \frac{\sum_{t\in I} \mathbb{P}^\sigma[\Omega_N \cap A_t = a \cap S_{t+1} = s]}{\sum_{t\in I} \mathbb{P}^\sigma[\Omega_N \cap A_t = a]} = (*)$$

Denoting

$$i\text{-Tries-In-}I = \{\omega \in \Omega_N \mid |\{t \in I \mid A_t = a\}| = i\}$$
$$i\text{-Successes-In-}I = \{\omega \in \Omega_N \mid |\{t \in I \mid A_t = a \cap S_{t+1} = s\}| = i\}$$

we can rewrite the term $(*)$ by grouping runs with same "frequencies" as

$$(*) = \frac{\sum_{i=1}^{|I|} i \cdot \mathbb{P}^\sigma[i\text{-Successes-In-}I]}{\sum_{i=1}^{|I|} i \cdot \mathbb{P}^\sigma[i\text{-Tries-In-}I]} = (**)$$

Similarly to the previous lemma, we introduce runs with "success rate" higher and lower than $\delta(a)(s) + \varepsilon$, now relative to the indices of I. Formally,

$$High_i^I = i\text{-Tries-In-}I \cap \bigcup_{k > i\cdot\left(\delta(a)(s)+\varepsilon\right)} k\text{-Successes-In-}I$$
$$Normal_i^I = i\text{-Tries-In-}I \cap \bigcup_{k \le i\cdot\left(\delta(a)(s)+\varepsilon\right)} k\text{-Successes-In-}I$$

allows us to rewrite

$$(**) = \frac{\sum_{i=1}^{|I|}(i \cdot HighRate_i) \cdot \mathbb{P}^\sigma\left[High_i^I\right] + \sum_{i=1}^{|I|}(i \cdot NormalRate_i) \cdot \mathbb{P}^\sigma\left[Normal_i^I\right]}{\sum_{i=1}^{|I|} i \cdot \mathbb{P}^\sigma\left[High_i^I\right] + \sum_{i=1}^{|I|} i \cdot \mathbb{P}^\sigma\left[Normal_i^I\right]} = (***)$$

where each $HighRate_i \in (\delta(a)(s) + \varepsilon, 1]$ and $NormalRate_i \in [0, \delta(a)(s) + \varepsilon]$ are the average portions of "successes" among the "tries" in the respective $High_i^I$ and $Normal_i^I$. Hence we can safely use the upper bounds to show

$$(***) \le \frac{1 \cdot \sum_{i=1}^{|I|} i \cdot \mathbb{P}^\sigma\left[High_i^I\right] + (\delta(a)(s) + \varepsilon) \cdot \sum_{i=1}^{|I|} i \cdot \mathbb{P}^\sigma\left[Normal_i^I\right]}{\sum_{i=1}^{|I|} i \cdot \mathbb{P}^\sigma\left[High_i^I\right] + \sum_{i=1}^{|I|} i \cdot \mathbb{P}^\sigma\left[Normal_i^I\right]} = (****)$$

Since $(****) \ge (*) \ge \delta(a)(s) + 2\varepsilon$, we get by the same computation as for obtaining (4.2)

$$\sum_{i=1}^{|I|} i \cdot \mathbb{P}^\sigma\left[Normal_i^I\right] \le \frac{1-\delta-2\varepsilon}{\varepsilon} \cdot \sum_{i=1}^{|I|} i \cdot \mathbb{P}^\sigma\left[High_i^I\right] \tag{4.3}$$

for every finite $I \subseteq T$.

Second step. Now we consider particular I's leading to a contradiction. Let T be written as $\{t_1, t_2, \ldots\}$ so that $t_1 < t_2 < \cdots$. For $m < n$, we consider finite subsets $I_m^n = \{t_m, t_{m+1}, \ldots, t_n\}$ of T and will prove that

$$\lim_{m\to\infty} \lim_{n\to\infty} \sum_{i=1}^{|I_m^n|} i \cdot \mathbb{P}^\sigma\left[High_i^{I_m^n}\right] = 0 \tag{4.4}$$

As a consequence of (4.3) we obtain also $\lim_{m\to\infty}\lim_{n\to\infty}\sum_{i=1}^{|I_m^n|} i\cdot\mathbb{P}^\sigma\left[\mathit{Normal}_i^{I_m^n}\right]=0$ and thus $\lim_{m\to\infty}\lim_{n\to\infty}\sum_{i=1}^{|I_m^n|} i\cdot\mathbb{P}^\sigma[i\text{-Tries-In-}I_m^n]=0$, i.e. with growing m the average number of tries after m approaches 0, a contradiction with $\mathbb{P}^\sigma[A_t=a\mid\Omega_N]>\zeta$ for infinitely many t and $\mathbb{P}^\sigma[\Omega_N]>0$.

It remains to prove (4.4). Intuitively, we consider index sets that start later (at position $m\to\infty$) to avoid initial potentially large elements. Summands with high i's, i.e. runs with many tries, below denoted by $\mathcal{C}$, will be shown negligible by the central limit theorem (in the previous lemma the law of large numbers was sufficient). Further, we will have to argue that even summands with low i's are small for high enough m. This is due to the fact that either a is taken frequently enough on some runs ($\mathcal{A}$) or for high enough indices not any more on the other runs ($\mathcal{B}$).

Formally, let $\mathit{Inf}=\Omega_N\cap\{A_t=a$ for infinitely many $t\}$ and $\mathit{Fin}_{\geq k}=\Omega_N\cap\{A_t=a$ for only finitely many $t\}\cap\{A_t=a$ for some $t\geq k\}$. We split the sum $\sum_{i=1}^{|I_m^n|} i\cdot\mathbb{P}^\sigma\left[\mathit{High}_i^{I_m^n}\right]$ into

$$\underbrace{\sum_{i=1}^{\mathit{middle}(m)} i\cdot\mathbb{P}^\sigma\left[\mathit{High}_i^{I_m^n}\cap\mathit{Inf}\right]}_{\mathcal{A}}+\underbrace{\sum_{i=1}^{\mathit{middle}(m)} i\cdot\mathbb{P}^\sigma\left[\mathit{High}_i^{I_m^n}\cap\mathit{Fin}_{\geq m}\right]}_{\mathcal{B}}+\underbrace{\sum_{i=\mathit{middle}(m)+1}^{|I_m^n|} i\cdot\mathbb{P}^\sigma\left[\mathit{High}_i^{I_m^n}\right]}_{\mathcal{C}}$$

by defining an appropriate $\mathit{middle}:\mathbb{N}\to\mathbb{N}$. We show that each term approaches zero.

$\mathcal{A}$: Observe that for every i and m, we have $\lim_{n\to\infty}\mathbb{P}^\sigma[i\text{-Tries-In-}I_m^n\cap\mathit{Inf}]=0$. Hence also $\lim_{n\to\infty}\mathcal{A}=0$ for every m and irrespective of the choice of $\mathit{middle}(m)$, and thus $\lim_{m\to\infty}\lim_{n\to\infty}\mathcal{A}=0$.

$\mathcal{B}$: We define $\mathit{middle}(m)$ to be the largest number such that $\sum_{i=1}^{\mathit{middle}(m)} i\cdot\mathbb{P}^\sigma[\mathit{Fin}_{\geq m}]<1/m$. This trivially ensures $\lim_{m\to\infty}\mathcal{B}\leq\lim_{m\to\infty}1/m=0$.

$\mathcal{C}$: Since $\lim_{m\to\infty}\mathbb{P}^\sigma[\mathit{Fin}_{\geq m}]=0$, we obtain by the definition of middle that for $m\to\infty$ also $\mathit{middle}(m)\to\infty$. Consequently, it is sufficient to prove that

$$\lim_{n\to\infty}\sum_{i=k}^{|I_m^n|} i\cdot\mathbb{P}^\sigma\left[\mathit{High}_i^{I_m^n}\right]\to 0 \text{ for } k\to\infty \text{ uniformly for all } m\,. \tag{4.5}$$

Fix an arbitrary m. Let X_j denote the indicator random variable of the event that jth use of action a, when looking only at time points $t_m,t_{m+1},t_{m+2}\ldots$, resulted in the successor s. Precisely, let T_j be an auxiliary random variable with value t_ℓ such that $|\{q\mid m\leq q\leq\ell, A_{t_q}=a\}|=j$ and $A_{t_q}=a$; then X_j is 1 if $S_{T_j+1}=s$ and 0 otherwise. Due to the Markov property, X_j are Bernoulli i.i.d. with mean $\delta(a)(s)$. Further,

$$\mathit{High}_i^{I_m^n}\subseteq\left\{\frac{\sum_{j=1}^{i}X_j}{i}>\delta(a)(s)+\varepsilon\right\}$$

Therefore, by central limit theorem

$$\mathbb{P}^\sigma\left[\mathit{High}_i^{I}\right]\lessapprox\Phi(-\sqrt{i}\cdot\hat{\varepsilon})$$

where $\hat{\varepsilon}=\varepsilon/\sqrt{\delta(a)(s)\cdot(1-\delta(a)(s))}$ and Φ is the cumulative distribution function of the standard normal distribution and $\lessapprox$ denotes that the inequality $\leq$ holds "only for large i", i.e. in the limit. Consequently, for large k, we have

$$\lim_{n\to\infty} \sum_{i=k}^{|I_m^n|} i \cdot \mathbb{P}^\sigma\left[High_i^{I_m^n}\right] \lessapprox \sum_{i=k}^{\infty} i \cdot \Phi(-\sqrt{i} \cdot \hat{\varepsilon})$$

where the right-hand side does not depend on m and is thus a uniform bound for all m. Further, since $\Phi(-\sqrt{i} \cdot \hat{\varepsilon})$ decreases exponentially in $\sqrt{i}$, the right-hand side approaches 0 as $k \to 0$ (independently of m) and (4.5) follows. $\square$

Now we show, that Equation 4 is satisfied. For all $s \in S$ and $N \subseteq [n]$ such that $\mathbb{P}^\sigma[\Omega_N] = 0$, we have trivially

$$\sum_{a\in A} x_{a,N} \cdot \delta(a)(s) = \sum_{a\in Act(s)} x_{a,N}$$

and whenever $\mathbb{P}^\sigma[\Omega_N] > 0$ we have

$$\begin{aligned}
&\frac{1}{\mathbb{P}^\sigma[\Omega_N]} \sum_{a\in A} f_N(a) \cdot \delta(a)(s) \\
&= \frac{1}{\mathbb{P}^\sigma[\Omega_N]} \sum_{a\in A} \lim_{\ell\to\infty} \frac{1}{T_\ell} \sum_{t=1}^{T_\ell} \mathbb{P}^\sigma[A_t = a \mid \Omega_N] \cdot \mathbb{P}^\sigma[\Omega_N] \cdot \delta(a)(s) && \text{(definition of } f_N\text{)} \\
&= \sum_{a\in A} \lim_{\ell\to\infty} \frac{1}{T_\ell} \sum_{t=1}^{T_\ell} \mathbb{P}^\sigma[A_t = a \mid \Omega_N] \cdot \delta(a)(s) && \text{(linearity of the limit)} \\
&= \sum_{a\in A} \lim_{\ell\to\infty} \frac{1}{T_\ell} \sum_{t=1}^{T_\ell} \mathbb{P}^\sigma[A_t = a \mid \Omega_N] \cdot \Delta_t^N(a)(s) && \text{(Lemma 4.3)} \\
&= \lim_{\ell\to\infty} \frac{1}{T_\ell} \sum_{t=1}^{T_\ell} \sum_{a\in A} \mathbb{P}^\sigma[A_t = a \mid \Omega_N] \cdot \Delta_t^N(a)(s) && \text{(definition of } T_\ell\text{)} \\
&= \lim_{\ell\to\infty} \frac{1}{T_\ell} \sum_{t=1}^{T_\ell} \mathbb{P}^\sigma[S_{t+1} = s \mid \Omega_N] && \text{(definition of } \Delta_t^N\text{)} \\
&= \lim_{\ell\to\infty} \frac{1}{T_\ell} \sum_{t=1}^{T_\ell} \mathbb{P}^\sigma[S_t = s \mid \Omega_N] && \text{(reindexing and Cesaro limit)} \\
&= \lim_{\ell\to\infty} \frac{1}{T_\ell} \sum_{t=1}^{T_\ell} \sum_{a\in Act(s)} \mathbb{P}^\sigma[A_t = a \mid \Omega_N] && (s \text{ must be followed by } a \in Act(s)) \\
&= \frac{1}{\mathbb{P}^\sigma[\Omega_N]} \sum_{a\in Act(s)} \lim_{\ell\to\infty} \frac{1}{T_\ell} \sum_{t=1}^{T_\ell} \mathbb{P}^\sigma[A_t = a \mid \Omega_N] \cdot \mathbb{P}^\sigma[\Omega_N] && \text{(linearity of the limit)} \\
&= \frac{1}{\mathbb{P}^\sigma[\Omega_N]} \sum_{a\in Act(s)} f_N(a)\ . && \text{(definition of } f_N\text{)}
\end{aligned}$$

Equation 5. For all $i \in [n]$, we have

$$\sum_{N \subseteq [n]} \sum_{a \in A} x_{a,N} \cdot \boldsymbol{r}_i(a) \;\geq\; \mathbb{E}^\sigma[\mathrm{lr}_{\inf}(\boldsymbol{r}_i)] \geq \boldsymbol{exp}_i$$

where the second inequality is due to σ being a witness strategy and the first inequality follows from the following:

$$\begin{aligned}
&\sum_{N \subseteq [n]} \sum_{a \in A} x_{a,N} \cdot \boldsymbol{r}_i(a) \\
&= \sum_{\substack{N \subseteq [n] \\ \mathbb{P}^\sigma[\Omega_N] > 0}} \sum_{a \in A} f_N(a) \cdot \boldsymbol{r}_i(a) && \text{(definition of } x_{a,N}\text{)} \\
&= \sum_{\substack{N \subseteq [n] \\ \mathbb{P}^\sigma[\Omega_N] > 0}} \sum_{a \in A} \boldsymbol{r}_i(a) \cdot \lim_{\ell \to \infty} \frac{1}{T_\ell} \sum_{t=1}^{T_\ell} \mathbb{P}^\sigma[A_t = a \mid \Omega_N] \cdot \mathbb{P}^\sigma[\Omega_N] && \text{(definition of } f_N\text{)} \\
&= \sum_{\substack{N \subseteq [n] \\ \mathbb{P}^\sigma[\Omega_N] > 0}} \mathbb{P}^\sigma[\Omega_N] \cdot \lim_{\ell \to \infty} \frac{1}{T_\ell} \sum_{t=1}^{T_\ell} \sum_{a \in A} \boldsymbol{r}_i(a) \cdot \mathbb{P}^\sigma[A_t = a \mid \Omega_N] && \text{(linearity of the limit)} \\
&\geq \sum_{\substack{N \subseteq [n] \\ \mathbb{P}^\sigma[\Omega_N] > 0}} \mathbb{P}^\sigma[\Omega_N] \cdot \liminf_{T \to \infty} \frac{1}{T} \sum_{t=1}^{T} \sum_{a \in A} \boldsymbol{r}_i(a) \cdot \mathbb{P}^\sigma[A_t = a \mid \Omega_N] && \text{(definition of } \liminf\text{)} \\
&= \sum_{\substack{N \subseteq [n] \\ \mathbb{P}^\sigma[\Omega_N] > 0}} \mathbb{P}^\sigma[\Omega_N] \cdot \liminf_{T \to \infty} \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}^\sigma[\boldsymbol{r}_i(A_t) \mid \Omega_N] && \text{(definition of the expectation)} \\
&\geq \sum_{\substack{N \subseteq [n] \\ \mathbb{P}^\sigma[\Omega_N] > 0}} \mathbb{P}^\sigma[\Omega_N] \cdot \mathbb{E}^\sigma[\mathrm{lr}_{\inf}(\boldsymbol{r}_i) \mid \Omega_N] && \text{(Fatou's lemma)} \\
&= \mathbb{E}^\sigma[\mathrm{lr}_{\inf}(\boldsymbol{r}_i)] && (\Omega_N\text{'s partition } \mathsf{Runs})
\end{aligned}$$

Although Fatou's lemma (see, e.g. [Roy88, Chapter 4, Section 3]) requires the function $\boldsymbol{r}_i(A_t)$ be non-negative, we can replace it with the non-negative function $\boldsymbol{r}_i(A_t) - \min_{a \in A} \boldsymbol{r}_i(a)$ and add the subtracted constant afterwards.

In order to show that Equations 6 and 7 hold, we prove the following lemma. This lemma is further necessary when relating the x-variables to the transient flow in Equation 3 later.

Lemma 4.4. For $N \subseteq [n]$ and $C \in \mathsf{MEC}$, we have

$$\sum_{a \in C} x_{a,N} = \mathbb{P}^\sigma[\Omega_N \cap \Omega_C] \;.$$

Proof. The proof is trivial for the case with $\mathbb{P}^\sigma[\Omega_N] = 0$. Let us now assume $\mathbb{P}^\sigma[\Omega_N] > 0$:

$$\sum_{a \in C} x_{a,N}$$

$$= \lim_{\ell\to\infty} \frac{1}{T_\ell} \sum_{t=1}^{T_\ell} \sum_{a\in C} \mathbb{P}^\sigma[A_t = a \mid \Omega_N] \cdot \mathbb{P}^\sigma[\Omega_N] \qquad \text{(definition of } x_{a,N} \text{ and } T_\ell)$$

$$= \lim_{\ell\to\infty} \frac{1}{T_\ell} \sum_{t=1}^{T_\ell} \sum_{a\in C} \Big(\mathbb{P}^\sigma[A_t = a \mid \Omega_N \cap \Omega_C] \cdot \frac{\mathbb{P}^\sigma[\Omega_N \cap \Omega_C]}{\mathbb{P}^\sigma[\Omega_N]} +$$

$$\mathbb{P}^\sigma[A_t = a \mid \Omega_N \setminus \Omega_C] \cdot \frac{\mathbb{P}^\sigma[\Omega_N \setminus \Omega_C]}{\mathbb{P}^\sigma[\Omega_N]}\Big) \cdot \mathbb{P}^\sigma[\Omega_N] \qquad \text{(partitioning of } \mathsf{Runs})$$

$$= \lim_{\ell\to\infty} \frac{1}{T_\ell} \sum_{t=1}^{T_\ell} \sum_{a\in C} \mathbb{P}^\sigma[A_t = a \mid \Omega_N \cap \Omega_C] \cdot \mathbb{P}^\sigma[\Omega_N \cap \Omega_C]$$

$$\Big(\lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} \mathbb{P}^\sigma[A_t = a \mid \Omega_N \setminus \Omega_C] = 0 \text{ for } a \in C\Big)$$

$$= \mathbb{P}^\sigma[\Omega_N \cap \Omega_C] \cdot \lim_{\ell\to\infty} \frac{1}{T_\ell} \sum_{t=1}^{T_\ell} \sum_{a\in C} \mathbb{P}^\sigma[A_t = a \mid \Omega_N \cap \Omega_C] \qquad \text{(linearity of the limit)}$$

$$= \mathbb{P}^\sigma[\Omega_N \cap \Omega_C] \cdot \lim_{\ell\to\infty} \frac{1}{T_\ell} \sum_{t=1}^{T_\ell} \mathbb{P}^\sigma[A_t \in C \mid \Omega_N \cap \Omega_C]$$

(taking two different actions at time t are disjoint events)

$$= \mathbb{P}^\sigma[\Omega_N \cap \Omega_C] \qquad \text{(since } A_t \in C \text{ for all but finitely many } t \text{ on } \Omega_C\text{, see below)}$$

It remains to prove that the last limit is equal to 1. We have

$$1 \geq \lim_{\ell\to\infty} \frac{1}{T_\ell} \sum_{t=1}^{T_\ell} \mathbb{P}^\sigma[A_t \in C \mid \Omega_N \cap \Omega_C] = \lim_{\ell\to\infty} \frac{1}{T_\ell} \sum_{t=1}^{T_\ell} \mathbb{E}^\sigma\left[\sum_{a\in C} \mathbb{1}_a(A_t) \mid \Omega_N \cap \Omega_C\right]$$

which is by dominated convergence theorem equal to

$$\mathbb{E}^\sigma\left[\lim_{\ell\to\infty} \frac{1}{T_\ell} \sum_{t=1}^{T_\ell} \sum_{a\in C} \mathbb{1}_a(A_t) \mid \Omega_N \cap \Omega_C\right] = \mathbb{E}^\sigma[1] = 1$$

by definition of Ω_C. □

Equation 6. For all $C \in \mathsf{MEC}, N \subseteq [n], i \in N$

$$\sum_{a\in C} x_{a,N} \cdot \boldsymbol{r}_i(a) \geq \sum_{a\in C} x_{a,N} \cdot \boldsymbol{sat}_i$$

follows trivially for $\mathbb{P}^\sigma[\Omega_N] = 0$, and whenever $\mathbb{P}^\sigma[\Omega_N] > 0$ we have

$$\sum_{a\in C} x_{a,N} \cdot \boldsymbol{r}_i(a)$$

$$\geq \liminf_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} \sum_{a\in C} \boldsymbol{r}_i(a) \cdot \mathbb{P}^\sigma[A_t = a \mid \Omega_N] \cdot \mathbb{P}^\sigma[\Omega_N]$$

(as above for Eq. 5, by def. of $x_{a,N}, f_N$, linearity of lim, def. of lim inf)

$$= \liminf_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} \sum_{a\in C} \boldsymbol{r}_i(a) \cdot \mathbb{P}^\sigma[A_t = a \mid \Omega_N \cap \Omega_C] \cdot \mathbb{P}^\sigma[\Omega_N \cap \Omega_C]$$

(as above in Lemma 4.4, by partitioning Runs, now with additional factor $\boldsymbol{r}_i(a)$)

$$\geq \mathbb{P}^\sigma[\Omega_N \cap \Omega_C] \cdot \mathbb{E}^\sigma[\mathrm{lr}_{\inf}(\boldsymbol{r}_i) \mid \Omega_N \cap \Omega_C]$$

(as above for Eq. 5, by def. of expectation and Fatou's lemma)

$$\geq \mathbb{P}^\sigma[\Omega_N \cap \Omega_C] \cdot \boldsymbol{sat}_i \qquad \text{(by definition of } \Omega_N \text{ and } i \in N)$$

$$= \sum_{a\in C} x_{a,N} \cdot \boldsymbol{sat}_i \qquad \text{(by Lemma 4.4)}$$

Equation 7. For every $i \in [n]$, by assumption on the strategy σ

$$\sum_{N\subseteq[n]:i\in N} \mathbb{P}^\sigma[\Omega_N] = \mathbb{P}^\sigma[\omega \in \mathsf{Runs} \mid \mathrm{lr}_{\inf}(\boldsymbol{r})(\omega)_i \geq \boldsymbol{sat}_i] \geq \boldsymbol{pr}_i$$

and the first term actually equals

$$\begin{aligned}
\sum_{N\subseteq[n]:i\in N} \sum_{a\in A} x_{a,N} &= \sum_{N\subseteq[n]:i\in N} \sum_{C\in\mathsf{MEC}} \sum_{a\in C} x_{a,N} && \text{(by (4.1))}\\
&= \sum_{N\subseteq[n]:i\in N} \sum_{C\in\mathsf{MEC}} \mathbb{P}^\sigma[\Omega_N \cap \Omega_C] && \text{(by Lemma 4.4)}\\
&= \sum_{N\subseteq[n]:i\in N} \mathbb{P}^\sigma[\Omega_N] && \text{(Ω_C's partition almost all Runs)}
\end{aligned}$$

4.2. **Transient behaviour and Equations 1–3.** Now we set the values for y_χ, $\chi \in A \cup (S \times 2^{[n]})$, and prove that they satisfy Equations 1–3 of L when the values $f_N(a)$ are assigned to $x_{a,N}$. One could obtain the values y_χ using the methods of [Put94, Theorem 9.3.8], which requires the machinery of deviation matrices. Instead, we can first simplify the behaviour of σ in the transient part to memoryless using [BBC+14] and then obtain y_χ directly, like in [EKVY08], as expected numbers of taking actions. To this end, for a state s we define $\Diamond s$ to be the set of runs that contain s.

Similarly to [BBC+14, Proposition 4.2 and 4.5], we modify the MDP G into another MDP $\overline{G}$ as follows: For each $s \in S, N \subseteq [n]$, we add a new absorbing state $f_{s,N}$. The only available action for $f_{s,N}$ leads back to $f_{s,N}$ with probability 1. We also add a new action $a_{s,N}$ to every $s \in S$ for each $N \subseteq [n]$. The distribution associated with $a_{s,N}$ assigns probability 1 to $f_{s,N}$. Finally, we remove all unreachable states. The construction of [BBC+14] is the same but with only a single value used for N. We denote the copy of each state s of G in $\overline{G}$ by $\overline{s}$.

Lemma 4.5. There is a strategy $\overline{\sigma}$ in $\overline{G}$ such that for every $C \in \mathsf{MEC}$ and $N \subseteq [n]$,

$$\sum_{s\in C} \mathbb{P}^{\overline{\sigma}}_{\overline{s_0}}[\Diamond f_{s,N}] = \mathbb{P}^\sigma_{s_0}[\Omega_C \cap \Omega_N] \ .$$

Proof. First, we consider an MDP G' created from G in the same way as $\overline{G}$, but instead of $f_{s,N}$ for each $s \in S, N \subseteq [n]$, we only have a single f_s; similarly for actions a_s. As in [BBC+14, Lemma 4.6], we obtain a strategy σ' in G' such that $\sum_{s\in C} \mathbb{P}^{\sigma'}_{s'_0}[\Diamond f_s] = \mathbb{P}^\sigma_{s_0}[\Omega_C]$.

We modify σ' into $\overline{\sigma}$ as follows. It behaves as σ', but instead of taking action a_s with probability p, we take each action $a_{s,N}$ with probability $p \cdot \frac{\mathbb{P}^{\sigma}_{s_0}[\Omega_C \cap \Omega_N]}{\mathbb{P}^{\sigma}_{s_0}[\Omega_C]}$. (For $\mathbb{P}^{\sigma}_{s_0}[\Omega_C] = 0$, we define $\overline{\sigma}$ arbitrarily.) Then

$$\sum_{s \in C} \mathbb{P}^{\overline{\sigma}}_{\overline{s_0}}[\Diamond f_{s,N}] = \sum_{s \in C} \frac{\mathbb{P}^{\sigma}_{s_0}[\Omega_C \cap \Omega_N]}{\mathbb{P}^{\sigma}_{s_0}[\Omega_C]} \cdot \mathbb{P}^{\sigma'}_{s'_0}[\Diamond f_s] = \mathbb{P}^{\sigma}_{s_0}[\Omega_C \cap \Omega_N]$$

□

By [EKVY08, Theorem 3.2], there is a memoryless strategy $\overline{\sigma}$ satisfying the lemma above such that

$$y_a := \sum_{t=1}^{\infty} \mathbb{P}^{\overline{\sigma}}_{\overline{s}}[A_t = a] \qquad \text{(for actions } a \text{ preserved in } \overline{G})$$
$$y_{s,N} := \mathbb{P}^{\overline{\sigma}}_{\overline{s_0}}[\Diamond f_{s,N}]$$

are finite values satisfying Equations 1 and 2, and, moreover,

$$y_{s,N} \geq \sum_{s \in C} \mathbb{P}^{\overline{\sigma}}[\Diamond f_{s,N}]\,.$$

By Lemma 4.5 for each $C \in \mathsf{MEC}$ we thus have

$$\sum_{s \in C} y_{s,N} \geq \mathbb{P}^{\sigma}[\Omega_C \cap \Omega_N]$$

and summing up over all C and N we have

$$\sum_{N \subseteq [n]} \sum_{s \in S} y_{s,N} \geq \sum_{N \subseteq [n]} \mathbb{P}^{\sigma}[\Omega_N]$$

where the first term is 1 by Equation 2, the second term is 1 by partitioning of Runs, hence they are actually equal and thus

$$\sum_{s \in C} y_{s,N} = \mathbb{P}^{\sigma}[\Omega_C \cap \Omega_N] = \sum_{a \in C} x_{a,N}$$

where the last equality follows by Lemma 4.4, yielding Equation 3.

5. Proof of Theorem 3.1: Solution to L induces witness strategy

Now we proceed to the proof of the third point of Theorem 3.1. Let $x_{a,N}, y_a, y_{s,N}, s \in S, a \in A, N \subseteq [n]$ be a solution to the system L. We show how it effectively induces a witness strategy σ.

We start with the recurrent part. We prove that even if the flow of Equation 4 is "disconnected" we may still play the actions with the exact frequencies $x_{a,N}$ on almost all runs. To formalize the frequency of an action a on a run, recall $\mathbb{1}_a$ is the indicator function of a, i.e. $\mathbb{1}_a(a) = 1$ and $\mathbb{1}_a(b) = 0$ for $a \neq b \in A$. Then $\boldsymbol{Freq}_a = \mathrm{lr}_{\inf}(\mathbb{1}_a)$ defines a vector random variable, indexed by $a \in A$. For the moment, we focus on strongly connected MDPs, i.e. the whole MDP is a MEC, and with $N \subseteq [n]$ fixed.

Firstly, we construct a strategy for each "strongly connected" part of the solution $x_{a,N}$ and connect the parts, thus averaging the frequencies. This happens at a cost of a small error used for transiting between the strongly connected parts. Secondly, we eliminate this error as we let the transiting happen with measure vanishing over time.

5.1. **x-values and recurrent behaviour.** To begin with, we show that x-values describe the recurrent behaviour only:

Lemma 5.1. Let $x_{a,N}, a \in A, N \subseteq [n]$ be a non-negative solution to Equation 4 of system L. Then for any fixed N, $X_N := \{s, a \mid x_{a,N} > 0, a \in Act(s)\}$ is a union of end components.

In particular, $X_N \subseteq \bigcup \mathsf{MEC}$, and for every $a \in A \setminus \bigcup \mathsf{MEC}$ and $N \subseteq [n]$, we have $x_{a,N} = 0$.

Proof. Denoting $x_{s,N} := \sum_{a \in Act(s)} x_{a,N} = \sum_{a \in A} x_{a,N} \cdot \delta(a)(s)$ for each $s \in S$, we can write

$$X_N = \{a \mid x_{a,N} > 0\} \cup \{s \mid x_{s,N} > 0\}\,.$$

Firstly, we need to show that for all $a \in X_N$, whenever $\delta(a)(s') > 0$ then $s' \in X_N$. Since $x_{s',N} \geq x_{a,N} \cdot \delta(a)(s') > 0$, we have $s' \in X_N$.

Secondly, let there be a path from $\hat{s}$ to $\hat{t}$ in X_N. We need to show that there is a path from $\hat{t}$ to $\hat{s}$ in X_N. Assume the contrary and denote $T \subseteq X_N$ the set of states with no path to $\hat{s}$ in X_N; we assume $\hat{t} \in T$. We write the path from $\hat{s}$ to $\hat{t}$ as $\hat{s} \cdots s'bt' \cdots \hat{t}$ where $s' \in X_N \setminus T$ and $t' \in T$. Then $b \in Act(s')$ and $\delta(b)(t') > 0$. Consequently,

$$\begin{aligned}
\sum_{s \in X_N \setminus T} \sum_{a \in A} x_a \cdot \delta(a)(s) &= \sum_{s \in X_N \setminus T} \sum_{a \in Act(s)} x_a \quad \text{(by summming Equation 4 over } s \in X_N \setminus T\text{)} \\
&= \sum_{s \in X_N \setminus T} \sum_{a \in Act(s)} \sum_{\bar{s} \in X_N \setminus T} x_a \cdot \delta(a)(\bar{s}) + \sum_{s \in X_N \setminus T} \sum_{a \in Act(s)} \sum_{\bar{s} \in T} x_a \cdot \delta(a)(\bar{s}) \\
&\qquad \text{(case split over target states)} \\
&> \sum_{s \in X_N \setminus T} \sum_{a \in Act(s)} \sum_{\bar{s} \in X_N \setminus T} x_a \cdot \delta(a)(\bar{s}) \qquad \text{(by } \delta(b)(t') > 0\text{)} \\
&= \sum_{\bar{s} \in X_N \setminus T} \sum_{\substack{a \in Act(s): \\ s \in X_N \setminus T}} x_a \cdot \delta(a)(\bar{s}) \qquad \text{(rearranging)} \\
&= \sum_{\bar{s} \in X_N \setminus T} \sum_{a \in A} x_a \cdot \delta(a)(\bar{s}) \qquad \text{(see below)}
\end{aligned}$$

which is a contradiction. The last equality follows by definition of T: actions enabled in T cannot lead to $X_N \setminus T$ since from $X_N \setminus T$ there is always a path to $\hat{s}$ and from T there is no path to $\hat{s}$. $\square$

We thus start with the construction of the recurrent behaviour from x-values. For the moment, we restrict to strongly connected MDP and focus on Equation 4 for a particular fixed $N \subseteq [n]$. Note that for a fixed $N \subseteq [n]$ we have a system of equations equivalent to the form

$$\sum_{a \in A} x_a \cdot \delta(a)(s) = \sum_{a \in Act(s)} x_a \qquad \text{for each } s \in S. \tag{5.1}$$

We set out to prove Corollary 5.5. This crucial observation states that even if the flow of Equation 4 is "disconnected", we may still play the actions with the exact frequencies $x_{a,N}$ on almost all runs.

Firstly, we construct a strategy for each "strongly connected" part of the solution x_a (each end-component of X_N of Lemma 5.1).

Lemma 5.2. In a strongly connected MDP G, let $x_{a,N}, a \in A$ be a non-negative solution to Equation 4 of system L for a fixed $N \subseteq [n]$ and $\sum_{a \in A} x_{a,N} > 0$. It induces a memoryless strategy ζ such that for every BSCCs D of G^ζ, every $a \in D \cap A$, and almost all runs in D holds

$$\boldsymbol{Freq}_a = \frac{x_{a,N}}{\sum_{a \in D \cap A} x_{a,N}}$$

i.e. $\mathbb{P}^\zeta\left[\boldsymbol{Freq}_a = \frac{x_{a,N}}{\sum_{a \in D \cap A} x_{a,N}} \mid \Omega_D\right] = 1$. Moreover, if all $x_{a,N}$'s are positive then G^ζ is a BSCC and $\boldsymbol{Freq}_a$ is almost surely constant.

Proof. By [BBC14, Lemma 4.3] applied to Equation (5.1), we get a memoryless strategy ζ such that $\mathbb{E}^\zeta[\boldsymbol{Freq}_a \mid \Omega_D] = x_{a,N} / \sum_{a \in D \cap A} x_{a,N}$. Furthermore, by the ergodic theorem, $\boldsymbol{Freq}_a$ returns the same value for almost all runs in Ω_D, hence is equal to $\mathbb{E}^\zeta[\boldsymbol{Freq}_a \mid \Omega_D]$. Finally, if all $x_{a,N}$'s are positive then all actions of G are used. Consequently, since G is strongly connected, G^ζ is also strongly connected. □

Secondly, we connect the parts (more end components of Lemma 5.1 within one MEC) and thus average the frequencies. This happens at a cost of small error used for transiting between the strongly connected parts.

Lemma 5.3. In a strongly connected MDP, let $x_{a,N}, a \in A$ be a non-negative solution to Equation 4 of system L for a fixed $N \subseteq [n]$ and $\sum_{a \in A} x_{a,N} > 0$. For every $\varepsilon > 0$, there is a memoryless strategy ζ^ε such that for all $a \in A$ almost surely

$$\boldsymbol{Freq}_a > \frac{x_{a,N}}{\sum_{a \in A} x_{a,N}} - \varepsilon$$

Proof. We obtain ζ^ε by a suitable perturbation of the strategy ζ from previous lemma in such a way that all actions get positive probabilities and the frequencies of actions change only slightly, similarly as in [BBC+14, Proposition 5.1, Part 2].

There exists an arbitrarily small (strictly) positive solution x'_a of Equation (5.1). Indeed, it suffices to consider a strategy τ which always takes the uniform distribution over the actions in every state and then assign $\mathbb{E}^\tau[\boldsymbol{Freq}_a]/M$ to x'_a for sufficiently large M. As the system of Equations (5.1) is linear and homogeneous, assigning $x_{a,N} + x'_a$ to $x_{a,N}$ also solves this system (and thus Equation 4 as well) and all values are positive. Consequently, Lemma 5.2 gives us a memoryless strategy ζ^ε satisfying almost surely (with $\mathbb{P}^{\zeta^\varepsilon}$-probability 1)

$$\boldsymbol{Freq}_a = \frac{(x_{a,N} + x'_a)}{\sum_{a' \in A}\left(x_{a',N} + x'_{a'}\right)} .$$

We may safely assume that $\sum_{a \in A} x'_a \le \frac{\varepsilon}{1-\varepsilon} \cdot \sum_{a \in A} x_{a,N}$. Then almost surely

$$\begin{aligned}
\boldsymbol{Freq}_a &= \frac{x_{a,N} + x'_a}{\sum_{a \in A}(x_{a,N} + x'_a)} && \text{(by Lemma 5.2)}\\
&> \frac{x_{a,N}}{\sum_{a \in A} x_{a,N} + \sum_{a \in A} x'_a} && \text{(by } x'_a > 0\text{)}\\
&\ge \frac{x_{a,N}}{\sum_{a \in A} x_{a,N} + \frac{\varepsilon}{1-\varepsilon} \cdot \sum_{a \in A} x_{a,N}} && \text{(by } \textstyle\sum_{a \in A} x'_a \le \frac{\varepsilon}{1-\varepsilon} \cdot \sum_{a \in A} x_{a,N}\text{)}\\
&= \frac{x_{a,N}}{\frac{1}{1-\varepsilon} \cdot \sum_{a \in A} x_{a,N}} && \text{(rearranging)}
\end{aligned}$$

$$\begin{aligned}
&= \frac{x_{a,N}}{\sum_{a\in A} x_{a,N}} - \varepsilon \cdot \frac{x_{a,N}}{\sum_{a\in A} x_{a,N}} && \text{(rearranging)}\\
&\geq \frac{x_{a,N}}{\sum_{a\in A} x_{a,N}} - \varepsilon && \left(\text{by } \tfrac{x_{a,N}}{\sum_{a\in A} x_{a,N}} \leq 1\right)
\end{aligned}$$

□

Thirdly, we eliminate this error as we let the transiting (by x'_a) happen with probability vanishing over time.

Lemma 5.4. In a strongly connected MDP, let ξ_i be a sequence of strategies, each with $\boldsymbol{Freq} = \boldsymbol{f}^i$ almost surely, and such that $\lim_{i\to\infty} \boldsymbol{f}^i$ is well defined. Then there is Markov strategy ξ such that almost surely

$$\boldsymbol{Freq} = \lim_{i\to\infty} \boldsymbol{f}^i \,.$$

Proof. This proof very closely follows the computation in [BBC+14, Proposition 5.1, Part "Moreover"], but for general ξ_i.

Given $a \in A$, let $\mathit{lf}_a := \lim_{i\to\infty} \boldsymbol{f}^i_a$. By definition of limit and the assumption that $\boldsymbol{Freq}_a = \mathrm{lr}_{\mathrm{inf}}(\mathbb{1}_a)$ is almost surely equal to $\boldsymbol{f}^i_a$ for each ξ_i, there is a subsequence ξ_j of the sequence ξ_i such that $\mathbb{P}^{\xi_j}\left[\mathrm{lr}_{\mathrm{inf}}(\mathbb{1}_a) \geq \mathit{lf}_a - 2^{-j-1}\right] = 1$. Note that for every $j \in \mathbb{N}$ there is $\kappa_j \in \mathbb{N}$ such that for all $a \in A$ and $s \in S$ we get

$$\mathbb{P}^{\xi_j}\left[\inf_{T\geq\kappa_j} \frac{1}{T}\sum_{t=0}^{T} \mathbb{1}_a(A_t) \geq \mathit{lf}_a - 2^{-j}\right] \geq 1 - 2^{-j}.$$

Let us consider a sequence $n_0, n_1, \dots$ of numbers where $n_j \geq \kappa_j$ and

$$\frac{\sum_{k<j} n_k}{n_j} \leq 2^{-j} \tag{5.2}$$

$$\frac{\kappa_{j+1}}{n_j} \leq 2^{-j} \tag{5.3}$$

We define ξ to behave as ξ_1 for the first n_1 steps, then as ξ_2 for the next n_2 steps, etc. In general, denoting by N_j the sum $\sum_{k<j} n_k$, the strategy ξ behaves as ξ_j between the N_j-th step (inclusive) and N_{j+1}-th step (non-inclusive). Note that such strategy is a Markov strategy.

Let us give some intuition behind ξ. The numbers in the sequence $n_0, n_1, \dots$ grow rapidly so that after ξ_j is simulated for n_j steps, the part of the history when ξ_k for $k < j$ were simulated becomes relatively small and has only minor impact on the current average reward (this is ensured by the condition $\frac{\sum_{k<j} n_k}{n_j} \leq 2^{-j}$). This gives us that almost every run has infinitely many prefixes on which the average reward w.r.t. $\mathbb{1}_a$ is arbitrarily close to lf_a infinitely often. To get that lf_a is also the long-run average reward, one only needs to be careful when the strategy ξ ends behaving as ξ_j and starts behaving as ξ_{j+1}, because then up to the κ_{j+1} steps we have no guarantee that the average reward is close to lf_a. This part is taken care of by picking n_j so large that the contribution (to the average reward) of the n_j steps according to ξ_j prevails over fluctuations introduced by the first κ_{j+1} steps according to ξ_{j+1} (this is ensured by the condition $\frac{\kappa_{j+1}}{n_j} \leq 2^{-j}$).

Let us now prove the correctness of the definition of ξ formally. We prove that almost all runs ω of G^ξ satisfy

$$\liminf_{T\to\infty} \frac{1}{T}\sum_{t=0}^{T} \mathbb{1}_a(A_t(\omega)) \geq \mathit{lf}_a\,.$$

Denote by E_k the set of all runs $\omega = s_0a_0s_1a_1\cdots$ of G^ξ such that for some $\kappa_k \leq d \leq n_k$ we have

$$\frac{1}{d}\sum_{j=N_j}^{N_j+d-1} \mathbb{1}_a(a_k) \quad < \quad \mathit{lf}_a - 2^{-k}.$$

We have $\mathbb{P}^\xi[E_j] \leq 2^{-j}$ and thus $\sum_{j=1}^{\infty} \mathbb{P}^\xi[E_j] = \frac{1}{2} < \infty$ holds. By Borel-Cantelli lemma [Roy88], almost surely only finitely many of E_j take place. Thus, almost every run $\omega = s_0a_0s_1a_1\cdots$ of G^ξ satisfies the following: there is ℓ such that for all $j \geq \ell$ and all $\kappa_j \leq d \leq n_j$ we have that

$$\frac{1}{d}\sum_{k=N_j}^{N_j+d-1} \mathbb{1}_a(a_k) \quad \geq \quad \mathit{lf}_a - 2^{-j}\,. \tag{5.4}$$

Consider $T \in \mathbb{N}$ such that $N_j \leq T < N_{j+1}$ where $j > \ell$. Below, we prove the following inequality

$$\frac{1}{T}\sum_{t=0}^{T} \mathbb{1}_a(a_t) \quad \geq \quad (\mathit{lf}_a - 2^{1-j})(1-2^{1-j})\,. \tag{5.5}$$

Taking the limit of (5.5) where T (and thus also j) goes to ∞, we obtain

$$\boldsymbol{Freq}_a(\omega) = \liminf_{T\to\infty} \frac{1}{T}\sum_{t=0}^{T} \mathbb{1}_a(a_t) \geq \liminf_{j\to\infty} (\mathit{lf}_a - 2^{1-j})(1-2^{1-j}) = \mathit{lf}_a = \lim_{i\to\infty} \boldsymbol{f}_a^i$$

yielding the lemma. It remains to prove (5.5). First, note that

$$\frac{1}{T}\sum_{t=0}^{T} \mathbb{1}_a(a_t) \quad \geq \quad \frac{1}{T}\sum_{t=N_{j-1}}^{N_j-1} \mathbb{1}_a(a_t) + \frac{1}{T}\sum_{t=N_j}^{T} \mathbb{1}_a(a_t)$$

and that by (5.4)

$$\frac{1}{T}\sum_{t=N_{j-1}}^{N_j-1} \mathbb{1}_a(a_t) = \frac{1}{n_j}\sum_{t=N_{j-1}}^{N_j-1} \mathbb{1}_a(a_t)\cdot\frac{n_j}{T} \geq (\mathit{lf}_a - 2^{1-j})\frac{n_j}{T}$$

which gives

$$\frac{1}{T}\sum_{t=0}^{T} \mathbb{1}_a(a_t) \;\geq\; (\mathit{lf}_a - 2^{1-j})\frac{n_j}{T} \;+\; \frac{1}{T}\sum_{t=N_j}^{T} \mathbb{1}_a(a_t). \tag{5.6}$$

Now, we distinguish two cases. First, if $T - N_j \leq \kappa_{j+1}$, then

$$\frac{n_j}{T} \geq \frac{n_j}{N_j+\kappa_{j+1}} = \frac{n_j}{N_{j-1}+n_j+\kappa_{j+1}} = 1 - \frac{N_{j-1}+\kappa_{j+1}}{N_{j-1}+n_j+\kappa_{j+1}} \geq (1-2^{1-j})$$

by (5.2) and (5.3). Therefore, by (5.6),

$$\frac{1}{T}\sum_{t=0}^{T} \mathbb{1}_a(a_t) \quad \geq \quad (\mathit{lf}_a - 2^{1-j})(1-2^{1-j}).$$

Second, if $T - N_j \geq \kappa_{j+1}$, then

$$\begin{aligned}
\frac{1}{T}\sum_{t=N_j}^{T} \mathbb{1}_a(a_t) &= \frac{1}{T-N_j+1}\sum_{t=N_j}^{T} \mathbb{1}_a(a_t) \cdot \frac{T-N_j+1}{T} \\
&\geq (\mathit{lf}_a - 2^{-j})\left(1 - \frac{N_{j-1}+n_j}{T}\right) && \text{(by (5.4))} \\
&\geq (\mathit{lf}_a - 2^{-j})\left(1 - 2^{-j} - \frac{n_j}{T}\right) && \text{(by (5.2))}
\end{aligned}$$

and thus, by (5.6),

$$\begin{aligned}
\frac{1}{T}\sum_{t=0}^{T} \mathbb{1}_a(a_t) &\geq (\mathit{lf}_a - 2^{1-j})\frac{n_j}{T} + (\mathit{lf}_a - 2^{-j+1})\left(1 - 2^{-j} - \frac{n_j}{T}\right) \\
&\geq (\mathit{lf}_a - 2^{1-j})\left(\frac{n_j}{T} + \left(1 - 2^{-j} - \frac{n_j}{T}\right)\right) \\
&\geq (\mathit{lf}_a - 2^{1-j})(1 - 2^{1-j})
\end{aligned}$$

which finishes the proof of (5.5). □

Now we know that strategies within an end component can be merged into a strategy with frequencies corresponding to the solution of Equation 4 for each fixed N.

Corollary 5.5. For a strongly connected MDP, let $x_{a,N}, a \in A$ be a non-negative solution to Equation 4 of system L for a fixed $N \subseteq [n]$ and $\sum_{a\in A} x_{a,N} > 0$. Then there is Markov strategy ξ_N such that for each $a \in A$ almost surely

$$\boldsymbol{Freq}_a = \frac{x_{a,N}}{\sum_{a\in A} x_{a,N}}.$$

Proof. The strategy ξ_N is constructed by Lemma 5.4 taking ξ_i to be $\zeta^{1/i}$ from Lemma 5.3. □

Remark 5.6. Note that using such strategy, all actions and states in the single MEC are visited infinitely often. (This will be later useful for the strategy complexity analysis.)

Since the fraction is independent of the initial state of the MDP, the frequency is almost surely the same also for all initial states. The reward of ξ_N is almost surely

$$\mathrm{lr}_{\mathrm{inf}}(\boldsymbol{r})(\omega) = \frac{\sum_a x_{a,N} \cdot \boldsymbol{r}(a)}{\sum_a x_{a,N}}.$$

When the MDP is not strongly connected, we obtain such ξ_N in each MEC C with $\sum_{a\in C} x_{a,N} > 0$ and the respective reward of almost all runs in C is thus

$$\mathbb{E}^{\xi_N}[\mathrm{lr}_{\mathrm{inf}}(\boldsymbol{r}) \mid \Omega_C] = \frac{\sum_{a\in C\cap A} x_{a,N} \cdot \boldsymbol{r}(a)}{\sum_{a\in C\cap A} x_{a,N}}. \tag{5.7}$$

Moreover, the long-run average reward is the same for almost all runs, which is a stronger property than in [BBC+14, Lemma 4.3], which does not hold for the induced strategy there. We need this property here in order to combine the satisfaction requirements.

$$\mathbb{P}^{\xi_N}\left[\mathrm{lr}_{\mathrm{inf}}(\boldsymbol{r}) = \frac{\sum_{a\in C\cap A} x_{a,N} \cdot \boldsymbol{r}(a)}{\sum_{a\in C\cap A} x_{a,N}} \mid \Omega_C\right] = 1. \tag{5.8}$$

5.2. **y-values and transient behaviour.** We now consider the transient part of the solution that plays ξ_N's with various probabilities. Let "switch to ξ_N in C" denote the event that a strategy updates its memory, while in C, into such an element that it starts playing exactly as ξ_N. We can stitch all ξ_N's together as follows:

Lemma 5.7. Let $\xi_N, N \subseteq [n]$ be strategies. Then every non-negative solution $y_a, y_{s,N}$, $a \in A, s \in S, N \subseteq [n]$ to Equation 1 effectively induces a strategy σ such that

$$\mathbb{P}^\sigma[\text{switch to } \xi_N \text{ in } s] = y_{s,N}$$

and σ is memoryless before the switch.

Proof. The idea is similar to [BBC+14, Proposition 4.2, Step 1]. However, instead of switching in s to ξ with some probability p, here we have to branch this decision and switch to ξ_N with probability $p \cdot \frac{y_{s,N}}{\sum_{N\subseteq[n]} y_{s,N}}$.

Formally, for every MEC C of G, we denote the number $\sum_{s\in C}\sum_{N\subseteq[n]} y_{s,N}$ by y_C. According to the Lemma 4.4 of [BBC+14] we have a stochastic-update strategy ϑ which stays eventually in each MEC C with probability y_C.

Then the strategy $\overline{\sigma}$ works as follows. It plays according to ϑ until a BSCC of G^ϑ is reached. This means that every possible continuation of the path stays in the current MEC C of G. Assume that C has states $s_1, \ldots, s_k$. At this point, the strategy $\overline{\sigma}$ changes its behaviour as follows: First, it strives to reach s_1 with probability one. Upon reaching s_1, it chooses randomly with probability $\frac{y_{s_1,N}}{y_C}$ to behave as ξ_N forever, or otherwise to follow on to s_2. If the strategy $\overline{\sigma}$ chooses to go on to s_2, it strives to reach s_2 with probability one. Upon reaching s_2, it chooses with probability $\frac{y_{s_2,N}}{y_C - \sum_{N\subseteq[n]} y_{s_1,N}}$ to behave as ξ_N forever, or to follow on to s_3, and so on, till s_k. That is, the probability of switching to ξ_N in s_i is

$$\frac{y_{s_i,N}}{y_C - \sum_{j=1}^{i-1}\sum_{N\subseteq[n]} y_{s_j,N}}.$$

Since ϑ stays in a MEC C with probability y_C, the probability that the strategy $\overline{\sigma}$ switches to ξ_N in s_i is equal to $y_{s_i,N}$. Further, as in [BBC+14] we can transform the part of $\overline{\sigma}$ before switching to ξ_N to a memoryless strategy and thus get strategy σ. □

Corollary 5.8. Let $\xi_N, N \subseteq [n]$ be strategies. Then every non-negative solution $y_a, y_{s,N}, x_{a,N}, a \in A, s \in S, N \subseteq [n]$ to Equations 1 and 3 effectively induces a strategy σ such that for every MEC C

$$\mathbb{P}^\sigma[\text{switch to } \xi_N \text{ in } C] = \sum_{a\in C\cap A} x_{a,N}$$

and σ is memoryless before the switch.

Proof. By Lemma 5.7 and Equation 3. □

5.3. **Proof of witnessing.** We now prove that the strategy σ of Corollary 5.8 with $\xi_N, N \subseteq [n]$ of Corollary 5.5 is indeed a witness strategy. Note that existence of ξ_N's depends on the sums of x-values being positive. This follows by Equation 2 and 3. We evaluate the strategy σ as follows:

$\mathbb{E}^\sigma[\mathrm{lr}_{\mathrm{inf}}(\boldsymbol{r})]$

$$= \sum_{C\in\mathsf{MEC}} \sum_{N\subseteq[n]} \mathbb{P}^{\sigma}[\text{switch to } \xi_N \text{ in } C] \cdot \mathbb{E}^{\xi_N}[\mathrm{lr}_{\inf}(\boldsymbol{r}) \mid \Omega_C]$$

$$\text{(by Equation 2, } \sum_{N\subseteq[n]} \mathbb{P}^{\sigma}[\text{switch to } \xi_N] = 1\text{)}$$

$$= \sum_{C\in\mathsf{MEC}} \sum_{N\subseteq[n]} \Big(\sum_{a\in C\cap A} x_{a,N}\Big) \cdot \mathbb{E}^{\xi_N}[\mathrm{lr}_{\inf}(\boldsymbol{r}) \mid \Omega_C] \qquad \text{(by Corollary 5.8)}$$

$$= \sum_{C\in\mathsf{MEC}} \sum_{\substack{N\subseteq[n]:\\ \sum_{a\in C\cap A} x_{a,N}>0}} \Big(\sum_{a\in C\cap A} x_{a,N}\Big) \cdot \Big(\sum_{a\in C\cap A} x_{a,N} \cdot \boldsymbol{r}(a) / \sum_{a\in C\cap A} x_{a,N}\Big) \qquad \text{(by (5.7))}$$

$$= \sum_{N\subseteq[n]} \sum_{C\in\mathsf{MEC}} \sum_{a\in C\cap A} x_{a,N} \cdot \boldsymbol{r}(a)$$

$$= \sum_{N\subseteq[n]} \sum_{a\in A\cap\bigcup\mathsf{MEC}} x_{a,N} \cdot \boldsymbol{r}(a)$$

$$= \sum_{N\subseteq[n]} \sum_{a\in A} x_{a,N} \cdot \boldsymbol{r}(a) \qquad \text{(by Lemma 5.1)}$$

$$\geq \boldsymbol{exp} \qquad \text{(by Equation 5)}$$

and for each $i \in [n]$ we have

$$\mathbb{P}^{\sigma}[\mathrm{lr}_{\inf}(\boldsymbol{r})_i \geq \boldsymbol{sat}_i] =$$

$$\sum_{C\in\mathsf{MEC}} \sum_{N\subseteq[n]} \mathbb{P}^{\sigma}[\text{switch to } \xi_N \text{ in } C] \cdot \mathbb{P}^{\xi_N}[\mathrm{lr}_{\inf}(\boldsymbol{r})_i \geq \boldsymbol{sat}_i \mid \Omega_C]$$

$$\text{(by Equation 2, } \sum_{N\subseteq[n]} \mathbb{P}^{\sigma}[\text{switch to } \xi_N] = 1\text{)}$$

$$= \sum_{C\in\mathsf{MEC}} \sum_{N\subseteq[n]} \Big(\sum_{a\in C\cap A} x_{a,N}\Big) \cdot \mathbb{P}^{\xi_N}[\mathrm{lr}_{\inf}(\boldsymbol{r})_i \geq \boldsymbol{sat}_i \mid \Omega_C] \qquad \text{(by Corollary 5.8)}$$

$$= \sum_{C\in\mathsf{MEC}} \sum_{\substack{N\subseteq[n]:\\ \sum_{a\in C\cap A} x_{a,N}>0}} \Big(\sum_{a\in C\cap A} x_{a,N}\Big) \cdot \mathbb{P}^{\xi_N}\Big[\sum_{a\in C\cap A} x_{a,N} \cdot \boldsymbol{r}(a)_i / \sum_{a\in C\cap A} x_{a,N} \geq \boldsymbol{sat}_i\Big]$$

$$\text{(by (5.8))}$$

$$\geq \sum_{C\in\mathsf{MEC}} \sum_{\substack{i\in N\subseteq[n]:\\ \sum_{a\in C\cap A} x_{a,N}>0}} \Big(\sum_{a\in C\cap A} x_{a,N}\Big) \cdot \mathbb{P}^{\xi_N}\Big[\sum_{a\in C\cap A} x_{a,N} \cdot \boldsymbol{sat}_i / \sum_{a\in C\cap A} x_{a,N} \geq \boldsymbol{sat}_i\Big]$$

$$\text{(by Equation 6)}$$

$$= \sum_{i\in N\subseteq[n]} \sum_{C\in\mathsf{MEC}} \sum_{a\in C\cap A} x_{a,N}$$

$$= \sum_{i\in N\subseteq[n]} \sum_{a\in A\cap\bigcup\mathsf{MEC}} x_{a,N}$$

$$= \sum_{i\in N\subseteq[n]} \sum_{a\in A} x_{a,N} \qquad \text{(by Lemma 5.1)}$$

$\geq \boldsymbol{pr}_i$ (by Equation 7)

Remark 5.9. The proof of the corresponding claim for ε-witness strategies proceeds as above. We get that the strategy σ of Corollary 5.8 with $\zeta^{\varepsilon}_N, N \subseteq [n]$ of Lemma 5.3 is an ε-witness strategy. △

6. Algorithmic complexity

In this section, we discuss the solutions to and complexity of all the introduced problems.

6.1. Solution to (multi-quant-conjunctive).

As we have seen, there are $\mathcal{O}(|G| \cdot n) \cdot 2^n$ variables in the linear program L. By Theorem 3.1, the upper bound on the algorithmic time complexity is polynomial in the number of variables in system L. Hence, the realizability problem for **(multi-quant-conjunctive)** can be decided in time polynomial in $|G|$ and exponential in n.

6.2. Solution to (multi-quant-joint) and the special cases.

In order to decide **(multi-quant-joint)**, the only subset of runs to exceed the probability threshold is the set of runs with all long-run rewards exceeding their thresholds, i.e. $\Omega_{[n]}$ (introduced in Section 4.1). The remaining runs need not be partitioned and can be all considered to belong to $\Omega_\emptyset$ without violating any constraint. Intuitively, each $x_{a,\emptyset}$ now stands for the original sum $\sum_{N\subseteq[n]:N\neq[n]} x_{a,N}$; similarly for y-variables. Consequently, the only non-zero variables of L indexed by N satisfy $N = [n]$ or $N = \emptyset$. The remaining variables can be left out of the system.

Requiring all variables $y_a, y_{s,N}, x_{a,N}$ for $a \in A, s \in S, N \in \{\emptyset, [n]\}$ be non-negative, the program is the following:

(1) transient flow: for $s \in S$

$$\mathbb{1}_{s_0}(s) + \sum_{a\in A} y_a \cdot \delta(a)(s) = \sum_{a\in Act(s)} y_a + y_{s,\emptyset} + y_{s,[n]}$$

(2) almost-sure switching to recurrent behaviour:

$$\sum_{s\in C} y_{s,\emptyset} + y_{s,[n]} = 1$$

(3) probability of switching in a MEC is the frequency of using its actions: for $C \in \mathsf{MEC}$

$$\sum_{s\in C} y_{s,\emptyset} = \sum_{a\in C} x_{a,\emptyset}$$

$$\sum_{s\in C} y_{s,[n]} = \sum_{a\in C} x_{a,[n]}$$

(4) recurrent flow: for $s \in S$

$$\sum_{a\in A} x_{a,\emptyset} \cdot \delta(a)(s) = \sum_{a\in Act(s)} x_{a,\emptyset}$$

$$\sum_{a\in A} x_{a,[n]} \cdot \delta(a)(s) = \sum_{a\in Act(s)} x_{a,[n]}$$

(5) expected rewards:

$$\sum_{a\in A}\Big(x_{a,\emptyset}+x_{a,[n]}\Big)\cdot \boldsymbol{r}(a)\geq \boldsymbol{exp}$$

(6) commitment to satisfaction: for $C\in\mathsf{MEC}$ and $i\in[n]$

$$\sum_{a\in C}x_{a,[n]}\cdot \boldsymbol{r}(a)_i\geq\sum_{a\in C}x_{a,[n]}\cdot \boldsymbol{sat}_i$$

(7) satisfaction:

$$\sum_{a\in A}x_{a,[n]}\geq pr$$

Since there are now $\mathcal{O}(|G|\cdot n)$ variables, the problem as well as its special cases can be decided in polynomial time.

Similarly, for **(mono-quant)** it is sufficient to consider $N=[n]=\{1\}$ and $N=\emptyset$ only. Consequently, for **(multi-qual)** $N=[n]$, and for **(mono-qual)** $N=[n]=\{1\}$ are sufficient, thus the index N can be removed completely.

Theorem 6.1. The **(multi-quant-joint)** realizability problem (and thus also all its special cases) can be decided in time polynomial in $|G|$ and n. □

6.3. **Solution to (multi-quant-conjunctive-joint).** The linear program for this "combined" problem can be easily derived from the program L in Fig. 4 as follows.

The first step consists in splitting the recurrent flow into two parts, *yes* and *no* Requiring all variables be non-negative, the program is the following:

(1) transient flow: for $s\in S$

$$\mathbb{1}_{s_0}(s)+\sum_{a\in A}y_a\cdot\delta(a)(s)=\sum_{a\in Act(s)}y_a+\sum_{N\subseteq[n]}(y_{s,N,yes}+y_{s,N,no})$$

(2) almost-sure switching to recurrent behaviour:

$$\sum_{\substack{s\in C\in\mathsf{MEC}\\ N\subseteq[n]}}(y_{s,N,yes}+y_{s,N,no})=1$$

(3) probability of switching in a MEC is the frequency of using its actions: for $C\in\mathsf{MEC}, N\subseteq[n]$

$$\sum_{s\in C}y_{s,N,yes}=\sum_{a\in C}x_{a,N,yes}$$

$$\sum_{s\in C}y_{s,N,no}=\sum_{a\in C}x_{a,N,no}$$

(4) recurrent flow: for $s\in S, N\subseteq[n]$

$$\sum_{a\in A}x_{a,N,yes}\cdot\delta(a)(s)=\sum_{a\in Act(s)}x_{a,N,yes}$$

$$\sum_{a\in A}x_{a,N,no}\cdot\delta(a)(s)=\sum_{a\in Act(s)}x_{a,N,no}$$

(5) expected rewards:

$$\sum_{\substack{a\in A,\\ N\subseteq[n]}} (x_{a,N,yes} + x_{a,N,no}) \cdot \boldsymbol{r}(a) \geq \boldsymbol{exp}$$

(6) commitment to satisfaction: for $C \in \mathsf{MEC}$, $N \subseteq [n]$, $i \in N$

$$\sum_{a\in C} x_{a,N,yes} \cdot \boldsymbol{r}(a)_i \geq \sum_{a\in C} x_{a,N,yes} \cdot \boldsymbol{sat}_i$$

$$\sum_{a\in C} x_{a,N,no} \cdot \boldsymbol{r}(a)_i \geq \sum_{a\in C} x_{a,N,no} \cdot \boldsymbol{sat}_i$$

(7) satisfaction: for $i \in [n]$

$$\sum_{\substack{a\in A,\\ N\subseteq[n]:i\in N}} x_{a,N,yes} + x_{a,N,no} \geq \boldsymbol{pr}_i$$

Note that this program has the same set of solutions as the original program, considering substitution $\alpha_{\beta,N} = \alpha_{\beta,N,yes} + \alpha_{\beta,N,no}$.

The second step consists in using the "*yes*" part of the flow for ensuring satisfaction of the (joint-SAT) constraint. Formally, we add the following additional equations (of type 6 and 7, respectively):

($\widetilde{6}$)

$$\sum_{a\in C} x_{a,N,yes} \cdot \boldsymbol{r}(a)_i \geq \sum_{a\in C} x_{a,N,yes} \cdot \widetilde{\boldsymbol{sat}}_i \qquad \text{for } i \in [n] \text{ and } N \subseteq [n]$$

($\widetilde{7}$)

$$\sum_{\substack{a\in A\\ N\subseteq[n]}} x_{a,N,yes} \geq \widetilde{pr}$$

Note that the number of variables is double that for **(multi-quant-conjunctive)**. Therefore, the complexity remains essentially the same:

Corollary 6.2. The algorithmic complexity for the **(multi-quant-conjuctive-joint)** is polynomial in the size of the MDP and exponential in n. $\square$

Remark 6.3. The strategies for the case of **(multi-quant-conjunctive-joint)** are very similar to that of **(multi-quant-conjunctive)**. Indeed, the structure of the constructed (ε-)witness strategies is the same: the memoryless strategy for reaching the desired MECs is followed by a stochastic-update switch to strategies for the recurrent behaviour. The only difference is the following. (ε-)witness strategies for **(multi-quant-conjunctive)** switch to strategies ξ_N (or ζ^ε_N), each given by values of x-variables indexed by a fixed $N \subseteq [n]$. In contrast, strategies for **(multi-quant-conjunctive-joint)** switch to strategies $\xi_{N,b}$ (or $\zeta^\varepsilon_{N,b}$), each given by values of x-variables indexed by a fixed $N \subseteq [n]$ *and* $b \in \{yes, no\}$. $\triangle$

Furthermore, we can also allow multiple constraints, i.e. more (joint-SAT) constraints or more (conjunctive-SAT), thus specifying probability thresholds for more value thresholds for each reward. Then instead of subsets of $[n]$ as so far, we consider subsets of the set of all constraints. The number of variables is then exponential in the number of constraints rather than just in the dimension of the rewards.

6.4. **Hardness.** The **(multi-quant-conjunctive-joint)** problem is also of significant theoretical interest since we can also prove the following hardness result:

Theorem 6.4. The **(multi-quant-conjunctive-joint)** problem is NP-hard (even without the (EXP) constraint).

Proof. We proceed by reduction from SAT. Let φ be a formula with the set of clauses $C = \{c_1, \ldots, c_k\}$ over atomic propositions $Ap = \{a_1, \ldots, a_p\}$. We denote $\overline{Ap} = \{\overline{a_1}, \ldots, \overline{a_p}\}$ the literals that are negations of the atomic propositions.

We define an MDP $G_\varphi = (S, A, Act, \delta, s_0)$ as follows:

- $S = \{s_i \mid i \in [p]\}$,
- $A = Ap \cup \overline{Ap}$,
- $Act(s_i) = \{a_i, \overline{a_i}\}$ for $i \in [p]$,
- $\delta(a_i)(s_{i+1}) = 1$ and $\delta(\overline{a_i})(s_{i+1}) = 1$ (actions are assigned Dirac distributions),
- $s_0 = s_1 = s_{p+1}$.

The constructed MDP is illustrated in Fig. 6. Intuitively, a run in G_φ repetitively chooses a valuation.

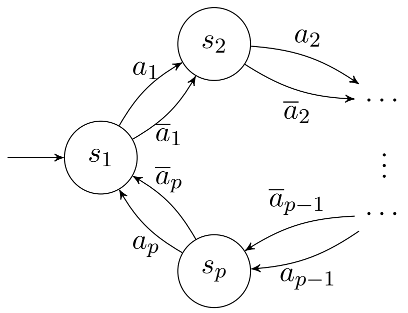


FIGURE 6. MDP G_φ

We define the dimension of the reward function to be $n = k + 2p$. We index the components of vectors with this dimension by $C \cup Ap \cup \overline{Ap}$. The reward function is defined for each $\ell \in A$ as follows:

- $\boldsymbol{r}(\ell)(c_i) = \begin{cases} 1 & \text{if } \ell \models c_i \\ 0 & \text{if } \ell \not\models c_i \end{cases}$
- $\boldsymbol{r}(\ell)(a_i) = \mathbb{1}_{a_i}$
- $\boldsymbol{r}(\ell)(\overline{a_i}) = \mathbb{1}_{\overline{a_i}}$

Intuitively, we get a positive reward for a clause when it is guaranteed to be satisfied by the choice of a literal. The latter two items simply count the number of uses of a literal; thus $\mathrm{lr}_{\inf}(\boldsymbol{r})_a = \boldsymbol{Freq}_a$.

The realizability problem instance R_φ is then defined by a conjunction of the following (conjunctive-SAT) and (joint-SAT) constraints:

$$\mathbb{P}^\sigma\left[\mathrm{lr}_{\inf}(\boldsymbol{r})_\ell \geq \frac{1}{p}\right] \geq \frac{1}{2} \qquad \text{for each } \ell \in Ap \cup \overline{Ap} \tag{conjunctive-S}$$

$$\mathbb{P}^\sigma\left[\bigwedge_{c \in C} \mathrm{lr}_{\inf}(\boldsymbol{r})_c \geq \frac{1}{p}\right] \geq \frac{1}{2} \tag{joint-S}$$

Intuitively, (conjunctive-S) ensures that almost all runs choose, for each atomic proposition, either the positive literal with frequency 1, or the negative literal with frequency 1; in other words, it ensures that the choice of valuation is consistent within the run almost surely. Indeed, since the choice between a_i and $\overline{a_i}$ happens every p steps, runs that mix both with positive frequency cannot exceed the value threshold $1/p$. Therefore, half of the runs must use only a_i, half must use only $\overline{a_i}$. Consequently, almost all runs choose one of them consistently.

Further, (joint-S) on the top ensures that there is a (consistent) valuation that satisfies all the clauses. Moreover, we require that this valuation is generated with probability at least 1/2. Actually, we only need probability strictly greater than 0.

We now prove that φ is satisfiable if and only if the problem instance defined above on MDP G_φ is realizable.

"Only if part": Let $\nu \subseteq Ap \cup \overline{Ap}$ be a satisfying valuation for φ. We define σ to have initial distribution on memory elements m_1, m_2 with probability 1/2 each. With memory m_1 we always choose action from ν and with memory m_2 from the "opposite valuation" $\overline{\nu}$ (where $\overline{\overline{a}}$ is identified with a).

Therefore, each literal has frequency $1/p$ either in the first or the second kind of runs. Further, the runs of the first kind (with memory m_1) satisfy all clauses.

"If part": Given a witness strategy σ for $R(\varphi)$, we construct a satisfying valuation. First, we focus on the property induced by the (conjunctive-S) constraint. We show that almost all runs uniquely induce a valuation

$$\nu_\sigma := \{\ell \in Ap \cup \overline{Ap} \mid \boldsymbol{Freq}_\ell > 0\}$$

which follows from the following lemma:

Lemma 6.5. For every witness strategy σ satisfying the (conjunctive-S) constraint, and for each $a \in Ap$, we have

$$\mathbb{P}^\sigma\left[\boldsymbol{Freq}_a = \frac{1}{p} \text{ and } \boldsymbol{Freq}_{\overline{a}} = 0\right] + \mathbb{P}^\sigma\left[\boldsymbol{Freq}_a = 0 \text{ and } \boldsymbol{Freq}_{\overline{a}} = \frac{1}{p}\right] = 1\,.$$

Proof. Let $a \in Ap$ be an arbitrary atomic proposition. To begin with, observe that due to the circular shape of MDP G_φ, we have

$$\boldsymbol{Freq}_a + \boldsymbol{Freq}_{\overline{a}} \leq 1/p \tag{6.1}$$

for every run. Indeed, $\boldsymbol{Freq}_a + \boldsymbol{Freq}_{\overline{a}} = \liminf_{T\to\infty} \frac{1}{T}\sum_{t=1}^{T} \mathbb{1}_a + \liminf_{T\to\infty} \frac{1}{T}\sum_{t=1}^{T} \mathbb{1}_{\overline{a}} \leq \liminf_{T\to\infty} \frac{1}{T}\sum_{t=1}^{T}(\mathbb{1}_a + \mathbb{1}_{\overline{a}}) = 1/p$.

Therefore, the two events $\boldsymbol{Freq}_a \geq 1/p$ and $\boldsymbol{Freq}_{\overline{a}} \geq 1/p$ are disjoint. Due to the (conjunctive-S) constraint, almost surely exactly one of the events occurs. Indeed,

$$1 \geq \mathbb{P}^\sigma\left[\boldsymbol{Freq}_a \geq \frac{1}{p} \cup \boldsymbol{Freq}_{\overline{a}} \geq \frac{1}{p}\right] = \mathbb{P}^\sigma\left[\boldsymbol{Freq}_a \geq \frac{1}{p}\right] + \mathbb{P}^\sigma\left[\boldsymbol{Freq}_{\overline{a}} \geq \frac{1}{p}\right] \geq \frac{1}{2} + \frac{1}{2} = 1$$

with the equality by disjointness of the events and the last inequality by (conjunctive-S).

Therefore, by (6.1), almost surely either $\boldsymbol{Freq}_a = 1/p$ and $\boldsymbol{Freq}_{\overline{a}} = 0$, or $\boldsymbol{Freq}_a = 0$ and $\boldsymbol{Freq}_{\overline{a}} = 1/p$. □

By the (joint-S) constraint, we have a set Ω_{sat}, with non-zero measure, of runs satisfying $\mathrm{lr}_{\inf}(\boldsymbol{r})_c \geq 1$ for each $c \in C$. By the previous lemma, almost all runs of Ω_{sat} induce unique valuations. Since there are finitely many valuation, at least one of them is induced by a set of non-zero measure. Let ω be one of the runs and ν the corresponding valuation. We claim that ν is a satisfying valuation for φ.

Let $c \in C$ be any clause, we show $\nu \models c$. Since $\mathrm{lr}_{\inf}(\boldsymbol{r})(\omega)_c \geq 1$, there is an action ℓ such that

- $\boldsymbol{Freq}_\ell(\omega) > 0$, and
- $\boldsymbol{r}(a)_\ell \geq 1$.

The former inequality implies that $\ell \in \nu$ and the latter that $\ell \models c$. Altogether, $\nu \models c$ for every $c \in C$, hence ν witnesses satisfiability of φ. □

Theorem 6.4 contrasts Theorem 6.1: while extension of (joint-SAT) with (EXP) can be solved in polynomial time, extending (joint-SAT) with (conjunctive-SAT) makes the problem NP-hard. Intuitively, adding (conjunctive-SAT) enforces us to consider the subsets of dimensions, and explains the exponential dependency on the number of dimensions in Theorem 3.1 (though our lower bound does not work for (conjunctive-SAT) with (EXP)).

The results are summarized in Table 2 and contrasted to the previously known polynomial bounds in Table 1.

7. Strategy complexity

First, we recall the structure of witness strategies generated from L in Section 5. In the first phase, a memoryless strategy is applied to reach MECs and switch to the recurrent strategies ξ_N. This switch is performed as a stochastic update, remembering the following two pieces of information: (1) the binary decision to stay in the current MEC C forever, and (2) the set $N \subseteq [n]$, such that almost all the produced runs belong to Ω_N. Each recurrent strategy ξ_N is then an infinite-memory strategy, where the memory is simply a counter. The counter determines which memoryless strategy ζ_N^ε is played.

7.1. Randomization and memory.

Similarly to the traditional setting with the expectation or the satisfaction semantics considered separately, the case with a single objective is simpler.

Lemma 7.1. Deterministic memoryless strategies are sufficient for witness strategies for **(mono-qual)**.

Proof. For each MEC, there is a value, which is the maximal long-run average reward. This is achievable for all runs in the MEC and using a memoryless strategy ξ. We prune the MDP to remove MECs with values below the threshold $\boldsymbol{sat}$. A witness strategy can be chosen to maximize the single long-run expected average objective, and thus also to be deterministic and memoryless [Put94]. Intuitively, in this case each MEC is either stayed at almost surely, or left almost surely if the value of the outgoing action is higher. □

Further, both for the expectation and the satisfaction semantics, deterministic memoryless strategies are sufficient for *quantitative* queries [FV97, BBE10] with single objective. In contrast, we show that both randomization and memory is necessary in our combined setting even for ε-witness strategies.

Example 7.2. Randomization and memory is necessary for **(mono-quant)** with $\boldsymbol{sat} = 1, \boldsymbol{exp} = 3, \boldsymbol{pr} = 0.55$ and the MDP and $\boldsymbol{r}$ depicted in Fig. 7. We have to remain in MEC $\{s, a\}$ with probability $p \in [0.1, 2/3]$, hence we need a randomized decision. Further, memoryless strategies would either never leave $\{s, a\}$ or would leave it eventually almost surely. Finally, the argument applies to ε-witness strategies, since the interval for p contains neither 0 nor 1 for sufficiently small ε.

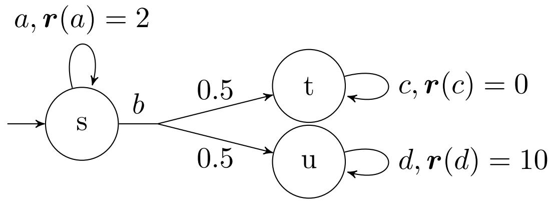

$a, \boldsymbol{r}(a) = 2$
s
b
0.5
0.5
t
u
$c, \boldsymbol{r}(c) = 0$
$d, \boldsymbol{r}(d) = 10$

FIGURE 7. An MDP with a single objective, where both randomization and memory is necessary

△

In the rest of the section, we discuss bounds on the size of the memory and the degree of randomization. Due to [BBC$^+$14, Section 5], infinite memory is indeed necessary for witnessing (joint-SAT) with $pr = 1$, hence also for **(multi-qual)**.

7.2. **Memory bounds for deterministic update.** We prove that finite memory is sufficient in several cases, namely for all ε-witness strategies and for **(mono-quant)** witness strategies. Moreover, these results also hold for deterministic-update strategies. Indeed, as one of our technical contributions, we prove that stochastic update at the moment of switching is not necessary and deterministic update is sufficient, requiring only a finite blow up in the memory size.

Lemma 7.3. Deterministic update is sufficient for witness strategies for **(multi-quant-conjuctive)** and **(multi-quant-joint)**. Moreover, finite memory is sufficient before switching to ξ_N's.

Proof idea. The stochastic decision during the switching in MEC C can be done as a deterministic update after a "toss", a random choice between two actions in C in one of the states of C. Such a toss does not affect the long-run average reward as it is only performed finitely many times.

More interestingly, in MECs where no toss is possible, we can remember which states were visited how many times and choose the respective probability of leaving or staying in C. □

Proof. Let σ be a strategy induced by L. We modify it into a strategy ϱ with the same distribution of the long-run average rewards. The only stochastic update that σ performs is in a MEC, switching to ξ_N with some probability. We modify σ into ϱ in each MEC C separately.

Tossing-MEC case First, we assume that there are $\mathit{toss}, a, b \in C$ with $a, b \in \mathit{Act}(\mathit{toss})$. Whenever σ should perform a step in $s \in C$ and possibly make a stochastic-update, say to m_1 with probability p_1 and m_2 with probability p_2, ϱ performs a "toss" instead. A (p_1, p_2)-toss consists of reaching *toss* with probability 1 (using a memoryless strategy), taking a, b with probabilities p_1, p_2, respectively, and making a deterministic update based on the result, in order to remember the result of the toss. After the toss, ϱ returns back to s with probability 1 (again using a memoryless strategy). Now as it already remembers the result of the (p_1, p_2)-toss, it changes the memory to m_1 or m_2 accordingly, by a deterministic update.

In general, since the stochastic-update probabilities depend on the action chosen and the state to be entered, we have to perform the toss for each combination before returning to s. Further, whenever there are more possible results for the memory update (e.g. various N), we can use binary encoding of the choices, say with k bits, and repeat the toss with the appropriate probabilities k-times before returning to s.

This can be implemented using finite memory. Indeed, since there are finitely many states in a MEC and σ is memoryless, there are only finitely many combinations of tosses to make and remember till the next simulated update of σ.

Tossfree-MEC case It remains to handle the case where, for each state $s \in C$, there is only one action $a \in \mathit{Act}(s) \cap C$. Then all strategies staying in C behave the same here, call this memoryless deterministic strategy ξ. Therefore, the only stochastic update that matters is to stay in C or not. The MEC C is left via each action a with the probability

$$\mathit{leave}_a := \sum_{t=1}^{\infty} \mathbb{P}^{\sigma}[S_t \in C \text{ and } A_t = a \text{ and } S_{t+1} \notin C]$$

and let $\{a \mid \mathit{leave}_a > 0\} = \{a_1, \ldots, a_\ell\}$ be the leaving actions. The strategy ϱ upon entering C performs the following. First, it leaves C via a_1 with probability leave_{a_1} (see below how), then via a_2 with probability $\frac{\mathit{leave}_{a_2}}{1-\mathit{leave}_{a_1}}$, and so on via a_i with probability

$$\frac{\mathit{leave}_{a_i}}{1 - \sum_{j=1}^{i-1} \mathit{leave}_{a_j}}$$

subsequently for each $i \in [\ell]$. After the last attempt with a_ℓ, if we are still in C, we update memory to stay in C forever (playing ξ).

Leaving C via a with probability *leave* can be done as follows. Let $\mathit{rate} = \sum_{s \notin C} \delta(a)(s)$ be the probability to actually leave C when taking a once. Then to achieve the overall probability *leave* of leaving we can reach s with $a \in \mathit{Act}(s)$ and play a with probability 1 and repeat this m times for some $m \in \mathbb{N}$ (if $\mathit{leave} = 1$ then $m = \infty$) and finally reach s once more and play a with probability $p \in [0, 1]$ and an action staying in C with the remaining probability. We now define m and p. If $\mathit{rate} = 1$ then $m = 0$ and $p = \mathit{leave}$. Assume $\mathit{rate} < 1$. Then we must ensure that the probability not to leave via a be

$$1 - \mathit{leave} = (1 - \mathit{rate})^m \cdot \big(p(1 - \mathit{rate}) + (1 - p)\big) \qquad (7.1)$$

Indeed, $(1 - \mathit{rate})^m$ stands for failing to leave m-times, and the last time we either choose a and fail again or not choose a at all. This requirement is equivalent to

$$m = \frac{\ln(1 - \mathit{leave}) - \ln(1 - p \cdot \mathit{rate})}{\ln(1 - \mathit{rate})}$$

For $p \in [0, 1]$ we have also $\frac{\ln(1 - p \cdot \mathit{rate})}{\ln(1 - \mathit{rate})} \in [0, 1]$. Therfore, in order to choose $m \in \mathbb{N}$, we can simply set $m := \lfloor \frac{\ln(1 - \mathit{leave})}{\ln(1 - \mathit{rate})} \rfloor$, which also ensures that $p \in [0, 1]$ for the respective $p := \frac{1}{\mathit{rate}}(1 - \frac{1 - \mathit{leave}}{(1 - \mathit{rate})^m})$, obtained from (7.1).

In order to implement the strategy in MECs of this second type, for each action it is sufficient to have a counter up to the respective m. □

Remark 7.4. Moreover, our proof also shows, that finite memory is sufficient before switching to ξ_N's (as defined in Section 5) for deterministic-update witnessing (and ε-witnessing) strategies. Therefore, finite memory deterministic update is sufficient for $\varepsilon-$witness strategies, in particular also for (joint-SAT), which improves the strategy complexity known from [BBC+14]. Note that in general, conversion of a stochastic-update strategy to a deterministic-update strategy requires an infinite blow up in the memory [dAHK07]. △

As a consequence, we obtain several bounds on memory size valid even for deterministic-update strategies. Firstly, infinite memory is required only for witness strategies:

Lemma 7.5. Deterministic-update with finite memory is sufficient for ε-witness strategies for **(multi-quant-conjuctive)** and **(multi-quant-joint)**.

Proof. After switching, memoryless strategies ζ_N^ε can be played instead of the sequence of $\zeta_N^{1/2^i}$. □

Remark 7.6. The previous proof of sufficiency of deterministic-update finite memory for ε-witness strategies applies also to **(multi-quant-conjunctive-joint)**. Indeed, firstly, Lemma 7.3 applies verbatim to **(multi-quant-conjunctive-joint)**. Secondly, we switch to only finitely many recurrent strategies due to Remark 6.3. △

Secondly, infinite memory is required only for multiple objectives:

Lemma 7.7. Deterministic-update strategies with finite memory are sufficient witness strategies for **(mono-quant)**.

Proof. After switching in a MEC C, we can play the following memoryless strategy. In C, there can be several components of the flow. We pick any with the largest long-run average reward. □

Further, the construction in the toss-free case gives us a hint for the respective lower bound on memory, even for the single-objective case.

Example 7.8. For deterministic-update ε-witness strategies for **(mono-quant)** problem, memory with size dependent on the transition probabilities is necessary. Indeed, consider the same realizability problem as in Example 7.2, but with a slightly modified MDP parametrized by λ, depicted in Fig. 8. Again, we have to remain in MEC $\{s, a\}$ with probability $p \in [0.1, 2/3]$. For ε-witness strategies the interval is slightly wider; let $\ell > 0$ denote the minimal probability with which any (ε-)witness strategy has to leave the MEC and all (ε-)witness strategies have to stay in the MEC with positive probability. We show that at least $\lceil\frac{\ell}{\lambda}\rceil$-memory is necessary. Observe that this setting also applies to the (EXP) setting of [BBC+14], e.g. $\boldsymbol{exp} = (0.5, 0.5)$ and the MDP of Fig. 9. Therefore, we provide a lower bound also for this simpler case (no MDP-dependent lower bound is provided in [BBC+14]).

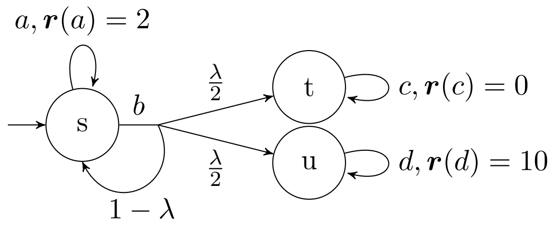

$a, \boldsymbol{r}(a) = 2$
s
b
$\frac{\lambda}{2}$
$\frac{\lambda}{2}$
t
u
$c, \boldsymbol{r}(c) = 0$
$d, \boldsymbol{r}(d) = 10$
$1-\lambda$

FIGURE 8. An MDP family with a single objective, where memory with size dependent on transition probabilities is necessary for deterministic-update strategies

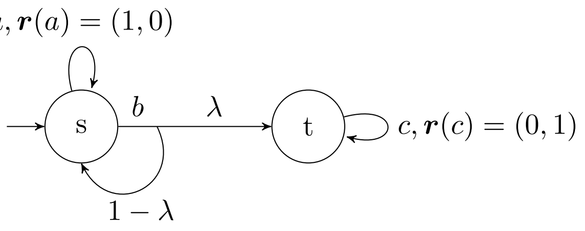

$a, \boldsymbol{r}(a) = (1, 0)$
s
b
λ
t
$c, \boldsymbol{r}(c) = (0, 1)$
$1-\lambda$

FIGURE 9. An MDP family, where memory with size dependent on transition probabilities is necessary for deterministic-update strategies even for (EXP) studied in [BBC+14]

For a contradiction, assume there are less than $\lceil\frac{\ell}{\lambda}\rceil$ memory elements. Then, by the pigeonhole principle, in the first $\lceil\frac{\ell}{\lambda} - 1\rceil$ visits of s, some memory element m appears twice. Note that due to the deterministic updating, each run generates the same play, thus the same sequence of memory elements. Let p be the probability to eventually leave s provided we are in s with memory m.

If $p = 0$ then the probability to leave s at the start is less than $\lceil\frac{\ell}{\lambda} - 2\rceil \cdot \lambda < \ell$, a contradiction. Indeed, we have at most $\lceil\frac{\ell}{\lambda} - 2\rceil$ tries to leave s before obtaining memory m and with every try we leave s with probability at most λ; we conclude by the union bound.

Let $p > 0$. Due to the deterministic updates, all runs staying in s use memory m infinitely often. Since $p > 0$, there is a finite number of steps such that (1) during these steps the overall probability to leave s is at least $p/2$ and (2) we are using m again. Consequently, the probability of the runs staying in s is 0, a contradiction. △

7.3. **Memory bounds for stochastic update.** Although we have shown that stochastic update is not necessary, it may be helpful when memory is small.

Lemma 7.9. Stochastic-update 2-memory strategies are sufficient for witness strategies for **(mono-quant)**.

Proof. The strategy σ of Section 5, which reaches the MECs and stays in them with given probability, is memoryless up to the point of switch by Corollary 5.8. Further, we can achieve the optimal value in each MEC using a memoryless strategy as in Lemma 7.7. $\square$

Theorem 7.10. Upper bounds on memory size for stochastic-update ε-witness strategies are as follows:

- **(multi-qual)** 2 memory elements,
- **(multi-quant-joint)** 3 memory elements,
- **(multi-quant-conjunctive)** $2^n + 1$ memory elements,
- **(multi-quant-conjunctive-joint)** $2^{n+1} + 1$ memory elements.

Proof. The structure of ε-witness strategies is described in Remark 5.9. Let us recall from Corollary 5.8 that strategy σ is memoryless before the switch. For **(multi-qual)**, **(multi-quant-joint)** and **(multi-quant-conjunctive)**, we perform the stochastic-update switch to different memory elements corresponding to the different strategies ζ_N^ε. From Lemma 5.3 we have that every such strategy ζ_N^ε is also memoryless. From Lemma 5.7 we have that we switch only to such ζ_N^ε for $N \subseteq [n]$, which correspond to possible nonzero variables $y_{s,N}$. Therefore, the number of memory elements needed is the number of possible nonzero variables $y_{s,N}$ for $N \subseteq [n]$ and additionally one element for the strategy σ before the switch.

Altogether, we get the following upper bounds on memory size of ε-witness strategies. For **(multi-quant-conjunctive)**, 2^n+1 memory elements are sufficient, since all of the $y_{s,N}$ for $N \subseteq [n]$ can be positive. For **(multi-quant-joint)**, 3 memory elements are sufficient, because we use only $y_{s,[n]}$ and $y_{s,\emptyset}$ as discussed in 6.2. Finally for **(multi-qual)**, 2 memory elements are sufficient, because we use only y_s as in 3.2.1.

Due to Remark 6.3, the bound on the number of recurrent strategies for **(multi-quant-conjunctive-joint)** is twice as large as for **(multi-quant-conjunctive)**, i.e., 2^{n+1}. The upper bound on the size of memory for ε-witness strategies for **(multi-quant-conjunctive-joint)** is thus $1 + 2^{n+1}$, compared to $1 + 2^n$ for **(multi-quant-conjunctive)**. $\square$

Example 7.11. For **(multi-quant-joint)**, ε-witness strategies may require memory with at least 3 elements. Consider an MDP with two states s and t with transitions and rewards as depicted in Fig. 10. Further, let $\boldsymbol{sat} = (1,0,0)$, $\boldsymbol{pr} = \frac{1}{2}$ and $\boldsymbol{exp} = (0,1,1)$.

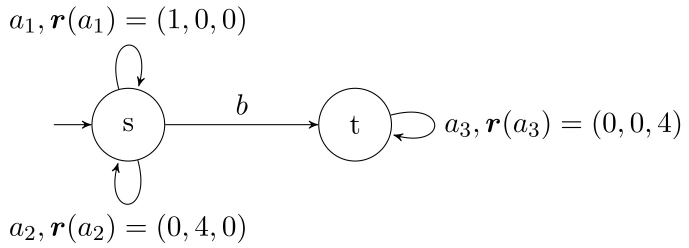

$a_1, \boldsymbol{r}(a_1) = (1,0,0)$
s
b
t
$a_3, \boldsymbol{r}(a_3) = (0,0,4)$
$a_2, \boldsymbol{r}(a_2) = (0,4,0)$

FIGURE 10. An MDP where 3-memory is necessary for **(multi-quant-joint)**

Suppose 2 memory elements are sufficient. In state s for each memory element we can either stay in s or go with some positive probability to state t. Therefore we have three cases on the behaviour in s regarding the transition to t:

(1) for each memory element we have positive probability p_1 and p_2 respectively, to go to state t,

(2) for both memory elements we have zero probability to go to t and
(3) for one memory element, say memory element 1, we have zero probability and for the other one, say memory element 2, we have positive probability p to go to t.

In the first case, we go to t eventually almost surely. Indeed, in each step we enter t with probability at least $\min(p_1, p_2)$ and cannot return back. Therefore, we stay in t forever and thus we cannot satisfy the satisfaction constraint.

In the second case, we never enter state t. Hence, we cannot satisfy the expectation constraint, because $\boldsymbol{r}(a_1)_3 = \boldsymbol{r}(a_2)_3 = 0$.

In the third case, we firstly assume that we switch from memory 1 to 2 with some positive probability p_1. Then in each step we have at least probability $p_1 \cdot p$ to enter t. Therefore, we end up in state t almost surely, not satisfying constraints, as shown above. Secondly, suppose we cannot switch from memory 1 to 2. Then we almost surely end up in state s with memory 1 or in state t. In state s with memory 1 we can either play action a_1 with probability 1 or with smaller potentially zero probability q. In the former case, $\mathrm{lr}(\boldsymbol{r}_2) = 0$, thus violating the expectation constraint. In the latter case, for almost every run $\mathrm{lr}(\boldsymbol{r}_1) \leq 1 - q$, contradicting the satisfaction constraint.

Note that a witnessing strategy exists, which uses only 3 memory elements. On half of the runs, we play only action a_1 to satisfy the satisfaction constraint. So we define $\sigma_n(s,1)(a_1) = 1$. To satisfy the expectation constraint for $\boldsymbol{r}_2$ we define $\sigma_n(s,2)(a_2) = 1$. With the last memory element we want to satisfy the expectation constraint for $\boldsymbol{r}_3$ and thus we define $\sigma_n(s,3)(b) = 1$ and $\sigma_n(t,3)(a_3) = 1$. We define the initial distribution by $\alpha(1) = \frac{1}{2}$, $\alpha(2) = \frac{1}{4}$ and $\alpha(3) = \frac{1}{4}$ and therefore the memory update function not to change memory. Consequently, the achieved expectation is $(\frac{1}{2} \cdot 1, \frac{1}{4} \cdot 4, \frac{1}{4} \cdot 4) \geq \boldsymbol{exp}$. △

However, even with stochastic update, the size of the finite memory cannot be bounded by a constant for **(multi-quant-conjunctive)**.

Example 7.12. Even ε-witness strategy for **(multi-quant-conjunctive)** may require memory with at least n memory elements. Consider an MDP with a single state s and self-loop a_i with reward $\boldsymbol{r}_i(a_j)$ equal to 1 for $i = j$ and 0 otherwise, for each $i \in [n]$. Fig. 11 illustrates the case with $n = 3$. Further, let $\boldsymbol{sat} = \mathbf{1}$ and $\boldsymbol{pr} = 1/n \cdot \mathbf{1}$.

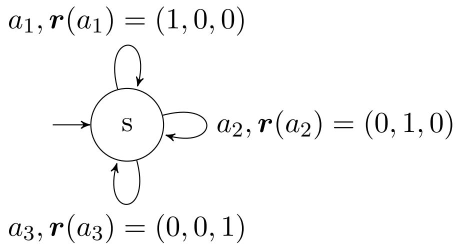

$a_1, \boldsymbol{r}(a_1) = (1,0,0)$
s
$a_2, \boldsymbol{r}(a_2) = (0,1,0)$
$a_3, \boldsymbol{r}(a_3) = (0,0,1)$

FIGURE 11. An MDP where n-memory is necessary, depicted for $n = 3$

The only way to ε-satisfy the constraints is that for each i, $1/n$ runs take only a_i, but for a negligible portion of time. Since these constraints are mutually incompatible for a single run, n different decisions have to be repetitively taken at s, showing the memory requirement. △

We summarize the upper and lower bounds for witness and ε-witness strategies in Table 3 and Table 4, respectively.

8. Pareto curve approximation and complexity summary

For a single objective, no Pareto curve is required and we can compute the optimal value of expectation in polynomial time by the linear program L with the objective function $\max \sum_{a\in A}(x_{a,\emptyset} + x_{a,\{1\}}) \cdot \boldsymbol{r}(a)$. For multiple objectives we obtain the following:

Theorem 8.1. For $\varepsilon > 0$, an ε-approximation of the Pareto curve for **(multi-quant-conjunctive-joint)** can be constructed in time polynomial in $|G|$ and $\frac{1}{\varepsilon}$ and exponential in n.

Proof. We replace $\boldsymbol{exp}$ in Equation 5 of L by a vector $\boldsymbol{v}$ of variables. Maximizing with respect to $\boldsymbol{v}$ is a multi-objective linear program. By [PY00], we can ε-approximate the Pareto curve in time polynomial in the size of the program and $\frac{1}{\varepsilon}$, and exponential in the number of objectives (dimension of $\boldsymbol{v}$). $\square$

The proof of Theorem 8.1 shows that we can obtain a Pareto-curve approximation also for possible values of the $\boldsymbol{sat}$ or $\boldsymbol{pr}$ vectors for a given $\boldsymbol{exp}$ vector. We simply replace these vectors by vectors of variables, obtaining a multi-objective linear program. If we want the complete Pareto-curve approximation for all the parameters $\boldsymbol{sat}$, $\boldsymbol{pr}$, and $\boldsymbol{exp}$, the number of objectives rises from n to $3 \cdot n$. The complexity is thus still polynomial in the size of the MDP and $1/\varepsilon$, and exponential in n.

In particular, for the single-objective case, we can compute also the optimal $\boldsymbol{pr}$ given $\boldsymbol{exp}$ and $\boldsymbol{sat}$, or the optimal $\boldsymbol{sat}$ given $\boldsymbol{pr}$ and $\boldsymbol{exp}$.

The complexity results are summarized in the following theorem:

Theorem 8.2. The algorithmic complexities are shown in Table 2. The bounds on the complexity of the witness and ε-witness strategies are as shown in Table 3 and Table 4, respectively.

Comments on the tables. **U:** denotes upper bounds (which suffice for all MDPs) and **L:** lower bounds (which are required in general for some MDPs). Results without reference are induced by the specialization or generalization relation depicted in Fig. 1 and for Table 3 and 4 by $\varepsilon-$witness strategies being a weaker notion than witness strategies. The abbreviations stoch.-up., det.-up., rand., det., inf., fin., and X-mem. stand for stochastic update, deterministic update, randomizing, deterministic, infinite-, finite- and X-memory strategies, respectively. Here n is the dimension of reward function and $p = 1/p_{\min}$ where $p_{\min}$ is the smallest positive probability in the MDP. Note that inf. actually means that the strategy is in form of a Markov strategy, see Section 5.

Remark 8.3. For a comparison, the results on previously studied subcases of our problems are depicted in Table 1. $\triangle$

Table 1. Previous results on algorithmic and strategy complexities. The abbreviations alg., strat., and c. stand for algorithmic, strategy, and complexity, respectively. Cases multiple and single refer to the number of objectives. Results for single-objective MDPs are based on classical literature, e.g. [Put94, Thm.9.1.8]. Results for MDPs with multiple objectives are due to [BBC+14].

Case	Alg. c.	Witness strat. c.	ε-witness strat. c.
multiple (joint-SAT)	$poly(\|G\|, n)$	**U:** det.-up. inf. **L:** rand. inf.	**U:** stoch.-up. 2-mem. **L:** rand. 2-mem.
multiple (EXP)	$poly(\|G\|, n)$	**U:** det.-up. inf. **L:** rand. inf.	**U:** stoch.-up. 2-mem., det.-up. fin. **L:** rand. 2-mem.
single (joint-SAT)	$poly(\|G\|)$	**U**=**L:** det. 1-mem.	**U**=**L:** det. 1-mem.
single (EXP)	$poly(\|G\|)$	**U**=**L:** det. 1-mem.	**U**=**L:** det. 1-mem.

Table 2. Algorithmic complexity results for each of the discussed cases.

Case	Algorithmic complexity
(multi-quant-conj.-joint)	$poly(\|G\|, 2^n)$ [Cor.6.2], NP-hard [Thm. 6.4]
(multi-quant-conj.)	$poly(\|G\|, 2^n)$ [Thm.3.1]
(multi-quant-joint)	$poly(\|G\|, n)$ [Thm.6.1]
(multi-qual)	$poly(\|G\|, n)$
(mono-quant)	$poly(\|G\|)$
(mono-qual)	$poly(\|G\|)$

Table 3. Witness strategy complexity bounds for each of the discussed cases.

Case	Witness strategy complexity
(multi-quant-conj.-joint)	**U:** det.-up. [Rem.7.6] inf. **L:** rand. inf.
(multi-quant-conj.)	**U:** det.-up. [Lem.7.3] inf. **L:** rand. inf.
(multi-quant-joint)	**U:** det.-up. inf. **L:** rand. inf.
(multi-qual)	**U:** det.-up. inf. **L:** rand. inf. [BBC+14, Sec.5]
(mono-quant)	**U:** stoch.-up. 2-mem. [Lem.7.9], det.-up. fin. [Lem.7.7] **L:** rand. 2-mem., for det.-up. p-mem.
(mono-qual)	**U:** (trivially also **L:**) det. 1-mem. [Lem.7.1]

9. Conclusion

We have presented a unifying solution framework to the expectation and satisfaction optimization of Markov decision processes with multiple objectives. This allows us to synthesize

TABLE 4. ε-witness strategy complexity bounds for each of the discussed cases.

Case	ε-witness strategy complexity
(multi-quant-conj.-joint)	**U:** stoch.-up. $(2^{n+1}+1)$-mem. [Thm.7.10], det.-up. fin. [Rem.7.6] **L:** rand. n-mem. [Ex.7.12], for det.-up. p-mem.
(multi-quant-conj.)	**U:** stoch.-up. $(2^{n}+1)$-mem. [Thm.7.10], det.-up. fin. [Lem.7.5] **L:** rand. n-mem. [Ex.7.12], for det.-up. p-mem.
(multi-quant-joint)	**U:** stoch.-up. 3-mem. [Thm.7.10], det.-up. fin. **L:** rand. 3-mem. [Ex.7.11]
(multi-qual)	**U:** stoch.-up. 2-mem. [Thm.7.10], det.-up. fin. **L:** rand. mem. [BBC+14, Sec.3]
(mono-quant)	**U:** stoch.-up. 2-mem., det.-up. fin. **L:** rand. [Ex.7.2] 2-mem. [Ex.7.2], for det.-up. p-mem. [Ex.7.8]
(mono-qual)	**U:** (trivially also **L:**) det. 1-mem.

optimal and ε-optimal risk-averse strategies. We have considered several possible combinations of the two semantics and provided algorithms for their solution as well as the complete picture of the complexities for all these cases.

Regarding the algorithmic complexity, we have shown that **(multi-quant-joint)** and all its special cases can be solved in polynomial time. For both **(multi-quant-conjunctive)** and **(multi-quant-conjunctive-joint)**, we have presented an algorithm that works in time polynomial in the size of MDP, but exponential in the dimension of reward function. However, the exponential in the dimension of reward function is not a limitation for most of practical purposes since the dimension is typically low. For the latter case we have also proved that the problem is NP-hard. The complexity of **(multi-quant-conjunctive)** remains an interesting open question. Moreover, our algorithms for Pareto-curve approximation work in time polynomial in the size of MDPs and exponential in the dimension of reward function. However, note that even for the special case of expectation semantics the current best known algorithms depend exponentially on the dimension of reward function [BBC+14].

We have also provided comprehensive results on strategy complexities. It is known that for both expectation and satisfaction semantics with single objective, deterministic memoryless strategies are sufficient [FV97, BBE10, BBC+14]. We have shown this carries over in the **(mono-qual)** case only. In contrast, for **(mono-quant)** both randomization and memory is necessary. However, we have also shown that only a restricted form of randomization (deterministic update) is necessary even for **(multi-quant)**, thus improving the upper bound for $\varepsilon-$witness strategies for the satisfaction problem of [BBC+14] to finite-memory deterministic update. Furthemore, we have established that with deterministic update the memory size is dependent on the MDP; the result also applies to the expectation problem of [BBC+14], where no MDP-dependent lower bound was given. We have presented upper bounds on stochastic update $\varepsilon-$witness strategies, which are constant for **(multi-qual)** and **(multi-quant-joint)**, and exponentially dependent on the dimension of reward function for **(multi-quant-conjunctive)** and **(multi-quant-conjunctive-joint)**. The question whether there are polynomially dependent upper bounds for the latter two cases stays open.

Acknowledgements We are very thankful to the anonymous reviewers for their helpful suggestions and pointing at gaps in the proofs of Lemma 4.3 and Lemma 5.1, and to Rasmus Ibsen-Jensen for discussing the proof of Lemma 4.2.

References

[Alt99] E. Altman. *Constrained Markov Decision Processes (Stochastic Modeling)*. Chapman & Hall/CRC, 1999.

[BBC$^+$14] T. Brázdil, V. Brožek, K. Chatterjee, V. Forejt, and A. Kučera. Markov decision processes with multiple long-run average objectives. *LMCS*, 10(1), 2014.

[BBE10] T. Brázdil, V. Brožek, and K. Etessami. One-counter stochastic games. In *FSTTCS*, pages 108–119, 2010.

[BCFK13] T. Brázdil, K. Chatterjee, V. Forejt, and A. Kučera. Trading performance for stability in Markov decision processes. In *LICS*, pages 331–340, 2013.

[BFRR14] V. Bruyère, E. Filiot, M. Randour, and J.-F. Raskin. Meet your expectations with guarantees: Beyond worst-case synthesis in quantitative games. In *STACS'14*, pages 199–213, 2014.

[BK08] C. Baier and J.-P. Katoen. *Principles of Model Checking*. MIT Press, 2008.

[CFW13] K. Chatterjee, V. Forejt, and D. Wojtczak. Multi-objective discounted reward verification in graphs and MDPs. In *LPAR'13*, pages 228–242, 2013.

[CH11] K. Chatterjee and M. Henzinger. Faster and dynamic algorithms for maximal end-component decomposition and related graph problems in probabilistic verification. In *SODA*, pages 1318–1336, 2011.

[CH12] K. Chatterjee and M. Henzinger. An $O(n^2)$ time algorithm for alternating Büchi games. In *SODA*, pages 1386–1399, 2012.

[CH14] K. Chatterjee and M. Henzinger. Efficient and dynamic algorithms for alternating Büchi games and maximal end-component decomposition. *JACM*, 2014.

[Cha07] K. Chatterjee. Markov decision processes with multiple long-run average objectives. In *FSTTCS*, pages 473–484, 2007.

[CL13] K. Chatterjee and J. Lacki. Faster algorithms for Markov decision processes with low treewidth. In *CAV*, pages 543–558, 2013.

[CMH06] K. Chatterjee, R. Majumdar, and T. A. Henzinger. Markov decision processes with multiple objectives. In *STACS*, pages 325–336, 2006.

[CR15] Lorenzo Clemente and Jean-François Raskin. Multidimensional beyond worst-case and almost-sure problems for mean-payoff objectives. In *LICS*, pages 257–268, 2015.

[CY95] C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *Journal of the ACM*, 42(4):857–907, 1995.

[CY98] C. Courcoubetis and M. Yannakakis. Markov decision processes and regular events. *Automatic Control, IEEE Transactions on*, 43(10):1399–1418, October 1998.

[dA97] L. de Alfaro. *Formal Verification of Probabilistic Systems*. PhD thesis, Stanford University, 1997.

[dAHK07] L. de Alfaro, T. A. Henzinger, and O. Kupferman. Concurrent reachability games. *Theor. Comput. Sci*, 386(3):188–217, 2007.

[DEKM98] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge Univ. Press, 1998.

[EKVY08] K. Etessami, M. Kwiatkowska, M. Vardi, and M. Yannakakis. Multi-objective model checking of Markov decision processes. *LMCS*, 4(4):1–21, 2008.

[FKN$^+$11] V. Forejt, M. Z. Kwiatkowska, G. Norman, D. Parker, and H. Qu. Quantitative multi-objective verification for probabilistic systems. In *TACAS*, pages 112–127, 2011.

[FKP12] V. Forejt, M. Z. Kwiatkowska, and D. Parker. Pareto curves for probabilistic model checking. In *ATVA'12*, pages 317–332, 2012.

[FKR95] J. A. Filar, D. Krass, and K. W Ross. Percentile performance criteria for limiting average Markov decision processes. *Automatic Control, IEEE Transactions on*, 40(1):2–10, Jan 1995.

[FV97] J. Filar and K. Vrieze. *Competitive Markov Decision Processes*. Springer-Verlag, 1997.

[How60] H. Howard. *Dynamic Programming and Markov Processes*. MIT Press, 1960.

[KGFP09] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas. Temporal-logic-based reactive mission and motion planning. *IEEE Transactions on Robotics*, 25(6):1370–1381, 2009.

[KNP02] M. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic symbolic model checker. In *TOOLS' 02*, pages 200–204, 2002.

[Kos88] J. Koski. Multicriteria truss optimization. In *Multicriteria Optimization in Engineering and in the Sciences*. 1988.

[Owe95] G. Owen. *Game Theory*. Academic Press, 1995.

[Put94] M.L. Puterman. *Markov Decision Processes*. John Wiley and Sons, 1994.

[PY00] C. H. Papadimitriou and M. Yannakakis. On the approximability of trade-offs and optimal access of web sources. In *FOCS*, pages 86–92, 2000.

[Roy88] H. Royden. *Real Analysis*. Prentice Hall, 3rd edition, 12 February 1988.

[RRS15] Mickael Randour, Jean-François Raskin, and Ocan Sankur. Percentile queries in multi-dimensional markov decision processes. In *CAV, Part I*, pages 123–139, 2015.

[Sch86] A. Schrijver. *Theory of Linear and Integer Programming*. Wiley-Interscience, 1986.

[SCK04] R. Szymanek, F. Catthoor, and K. Kuchcinski. Time-energy design space exploration for multi-layer memory architectures. In *DATE*, pages 318–323, 2004.

[Seg95] R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems*. PhD thesis, MIT, 1995.

[Var85] M. Vardi. Automatic verification of probabilistic concurrent finite state programs. In *FOCS*, pages 327–338, 1985.

[WL99] C. Wu and Y. Lin. Minimizing risk models in Markov decision processes with policies depending on target values. *Journal of Mathematical Analysis and Applications*, 231(1):47–67, 1999.

[YC03] P. Yang and F. Catthoor. Pareto-optimization-based run-time task scheduling for embedded systems. In *CODES+ISSS*, pages 120–125, 2003.

Appendix A. Limear program for the running example

(1) $1 + 0.5y_\ell = y_\ell + y_r + y_{s,\emptyset} + y_{s,\{1\}} + y_{s,\{2\}} + y_{s,\{1,2\}}$
$0.5y_\ell + y_a = y_a + y_{u,\emptyset} + y_{u,\{1\}} + y_{u,\{2\}} + y_{u,\{1,2\}}$
$y_r + y_b + y_e = y_b + y_c + y_{v,\emptyset} + y_{v,\{1\}} + y_{v,\{2\}} + y_{v,\{1,2\}}$
$y_c + y_d = y_d + y_e + y_{w,\emptyset} + y_{w,\{1\}} + y_{w,\{2\}} + y_{w,\{1,2\}}$

(2) $y_{u,\emptyset} + y_{u,\{1\}} + y_{u,\{2\}} + y_{u,\{1,2\}} + y_{v,\emptyset} + y_{v,\{1\}} + y_{v,\{2\}} + y_{v,\{1,2\}} + y_{w,\emptyset} + y_{w,\{1\}} + y_{w,\{2\}} + y_{w,\{1,2\}} = 1$

(3) $y_{u,\emptyset} = x_{a,\emptyset}$
$y_{u,\{1\}} = x_{a,\{1\}}$
$y_{u,\{2\}} = x_{a,\{2\}}$
$y_{u,\{1,2\}} = x_{a,\{1,2\}}$

$y_{v,\emptyset} + y_{w,\emptyset} = x_{b,\emptyset} + x_{c,\emptyset} + x_{d,\emptyset} + x_{e,\emptyset}$
$y_{v,\{1\}} + y_{w,\{1\}} = x_{b,\{1\}} + x_{c,\{1\}} + x_{d,\{1\}} + x_{e,\{1\}}$
$y_{v,\{2\}} + y_{w,\{2\}} = x_{b,\{2\}} + x_{c,\{2\}} + x_{d,\{2\}} + x_{e,\{2\}}$
$y_{v,\{1,2\}} + y_{w,\{1,2\}} = x_{b,\{1,2\}} + x_{c,\{1,2\}} + x_{d,\{1,2\}} + x_{e,\{1,2\}}$

(4) $0.5x_{\ell,\emptyset} = x_{\ell,\emptyset} + x_{r,\emptyset}$
$0.5x_{\ell,\{1\}} = x_{\ell,\{1\}} + x_{r,\{1\}}$
$0.5x_{\ell,\{2\}} = x_{\ell,\{2\}} + x_{r,\{2\}}$
$0.5x_{\ell,\{1,2\}} = x_{\ell,\{1,2\}} + x_{r,\{1,2\}}$

$0.5x_{\ell,\emptyset} + x_{a,\emptyset} = x_{a,\emptyset}$
$0.5x_{\ell,\{1\}} + x_{a,\{1\}} = x_{a,\{1\}}$

$0.5x_{\ell,\{2\}} + x_{a,\{2\}} = x_{a,\{2\}}$
$0.5x_{\ell,\{1,2\}} + x_{a,\{1,2\}} = x_{a,\{1,2\}}$

$x_{r,\emptyset} + x_{b,\emptyset} + x_{e,\emptyset} = x_{b,\emptyset} + x_{c,\emptyset}$
$x_{r,\{1\}} + x_{b,\{1\}} + x_{e,\{1\}} = x_{b,\{1\}} + x_{c,\{1\}}$
$x_{r,\{2\}} + x_{b,\{2\}} + x_{e,\{2\}} = x_{b,\{2\}} + x_{c,\{2\}}$
$x_{r,\{1,2\}} + x_{b,\{1,2\}} + x_{e,\{1,2\}} = x_{b,\{1,2\}} + x_{c,\{1,2\}}$

$x_{c,\emptyset} + x_{d,\emptyset} = x_{d,\emptyset} + x_{e,\emptyset}$
$x_{c,\{1\}} + x_{d,\{1\}} = x_{d,\{1\}} + x_{e,\{1\}}$
$x_{c,\{2\}} + x_{d,\{2\}} = x_{d,\{2\}} + x_{e,\{2\}}$
$x_{c,\{1,2\}} + x_{d,\{1,2\}} = x_{d,\{1,2\}} + x_{e,\{1,2\}}$

(5) $\boldsymbol{r}(\ell)x_{\ell,\emptyset} + \boldsymbol{r}(\ell)x_{\ell,\{1\}} + \boldsymbol{r}(\ell)x_{\ell,\{2\}} + \boldsymbol{r}(\ell)x_{\ell,\{1,2\}} + \boldsymbol{r}(r)x_{r,\emptyset} + \boldsymbol{r}(r)x_{r,\{1\}} + \boldsymbol{r}(r)x_{r,\{2\}} + \boldsymbol{r}(r)x_{r,\{1,2\}} + (4,0)x_{a,\emptyset} + (4,0)x_{a,\{1\}} + (4,0)x_{a,\{2\}} + (4,0)x_{a,\{1,2\}} + (1,0)x_{b,\emptyset} + (1,0)x_{b,\{1\}} + (1,0)x_{b,\{2\}} + (1,0)x_{b,\{1,2\}} + (0,0)x_{c,\emptyset} + (0,0)x_{c,\{1\}} + (0,0)x_{c,\{2\}} + (0,0)x_{c,\{1,2\}} + (0,1)x_{d,\emptyset} + (0,1)x_{d,\{1\}} + (0,1)x_{d,\{2\}} + (0,1)x_{d,\{1,2\}} + (0,0)x_{e,\emptyset} + (0,0)x_{e,\{1\}} + (0,0)x_{e,\{2\}} + (0,0)x_{e,\{1,2\}} \geq (1.1, 0.5)$

(6) $4x_{a,\{1\}} \geq 0.5x_{a,\{1\}}$
$0 \geq 0.5x_{a,\{2\}}$
$4x_{a,\{1,2\}} \geq 0.5x_{a,\{1,2\}}$
$0 \geq 0.5x_{a,\{1,2\}}$

$x_{b,\{1\}} \geq 0.5x_{b,\{1\}} + 0.5x_{c,\{1\}} + 0.5x_{d,\{1\}} + 0.5x_{e,\{1\}}$
$x_{d,\{2\}} \geq 0.5x_{b,\{2\}} + 0.5x_{c,\{2\}} + 0.5x_{d,\{2\}} + 0.5x_{e,\{2\}}$
$x_{b,\{1,2\}} \geq 0.5x_{b,\{1,2\}} + 0.5x_{c,\{1,2\}} + 0.5x_{d,\{1,2\}} + 0.5x_{e,\{1,2\}}$
$x_{d,\{1,2\}} \geq 0.5x_{b,\{1,2\}} + 0.5x_{c,\{1,2\}} + 0.5x_{d,\{1,2\}} + 0.5x_{e,\{1,2\}}$

(7) $x_{\ell,\{1\}} + x_{\ell,\{1,2\}} + x_{r,\{1\}} + x_{r,\{1,2\}} + x_{a,\{1\}} + x_{a,\{1,2\}} + x_{b,\{1\}} + x_{b,\{1,2\}} + x_{c,\{1\}} + x_{c,\{1,2\}} + x_{d,\{1\}} + x_{d,\{1,2\}} + x_{e,\{1\}} + x_{e,\{1,2\}} \geq 0.8$

$x_{\ell,\{2\}} + x_{\ell,\{1,2\}} + x_{r,\{2\}} + x_{r,\{1,2\}} + x_{a,\{2\}} + x_{a,\{1,2\}} + x_{b,\{2\}} + x_{b,\{1,2\}} + x_{c,\{2\}} + x_{c,\{1,2\}} + x_{d,\{2\}} + x_{d,\{1,2\}} + x_{e,\{2\}} + x_{e,\{1,2\}} \geq 0.8$

Faster Statistical Model Checking for Unbounded Temporal Properties

PRZEMYSŁAW DACA and THOMAS A. HENZINGER, IST Austria, Austria
JAN KŘETÍNSKÝ, Technische Universität München, Germany
TATJANA PETROV, IST Austria, Austria

We present a new algorithm for the statistical model checking of Markov chains with respect to unbounded temporal properties, including full linear temporal logic. The main idea is that we monitor each simulation run on the fly, in order to detect quickly if a bottom strongly connected component is entered with high probability, in which case the simulation run can be terminated early. As a result, our simulation runs are often much shorter than required by termination bounds that are computed a priori for a desired level of confidence on a large state space. In comparison to previous algorithms for statistical model checking our method is not only faster in many cases but also requires less information about the system, namely, only the minimum transition probability that occurs in the Markov chain. In addition, our method can be generalised to unbounded quantitative properties such as mean-payoff bounds.

CCS Concepts: •**Theory of computation** → **Logic and verification;** *Random walks and Markov chains; Modal and temporal logics;*

Additional Key Words and Phrases: Markov chains, mean payoff, simulation, statistical model checking, temporal logic

ACM Reference Format:
Przemysław Daca, Thomas A. Henzinger, Jan Křetínský, and Tatjana Petrov, 2016. Faster Statistical Model Checking for Unbounded Temporal Properties. *ACM Trans. Comput. Logic* 9, 4, Article 39 (March 2010), 26 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

Traditional numerical algorithms for the verification of Markov chains may be computationally intense or inapplicable, when facing a large state space or limited knowledge of the chain. To this end, statistical algorithms are used as a powerful alternative. *Statistical model checking* (SMC) typically refers to approaches where (i) finite paths of the Markov chain are sampled a finite number of times, (ii) the property of interest is verified for each sampled path (e.g. state r is reached), and (iii) hypothesis testing or statistical estimation is used to infer conclusions (e.g. state r is reached with probability at most 0.5) and give statistical guarantees (e.g. the conclusion is valid with 99% confidence). SMC approaches differ in (a) the class of properties they can verify (e.g. bounded or unbounded properties), (b) the strength of statistical guarantees they provide (e.g. confidence bounds, only asymptotic convergence of the method towards

This is an extended version of [Daca et al. 2016a] with full proofs and an extended discussion of the experimental results. This research was funded in part by the European Research Council (ERC) under grant agreement 267989 (QUAREM), the Austrian Science Fund (FWF) under grants project S11402-N23 (RiSE) and Z211-N23 (Wittgenstein Award), the People Programme (Marie Curie Actions) of the European Union's Seventh Framework Programme (FP7/2007-2013) REA Grant No 291734, the SNSF Advanced Postdoc Mobility Fellowship — grant number P300P2_161067, and the Czech Science Foundation under grant agreement P202/12/G061.

Table I. SMC approaches to Markov chain verification, organised by (i) the class of verifiable properties, and (ii) by the required information about the Markov chain, where p_{min} is the minimum transition probability, $|S|$ is the number of states, and λ is the second largest eigenvalue of the chain.

	no info	p_{min}	$\lvert S\rvert, p_{\mathsf{min}}$	λ	topology
LTL, mean payoff	$\times$	**here**	[Brázdil et al. 2014](LTL)		
$\Diamond, \mathbf{U}$	$\times$	**here**	—"—	[Younes et al. 2010]	[He et al. 2010] [Younes et al. 2010]
bounded	[Younes and Simmons 2002] [Sen et al. 2004]				

the correct value, or none), and (c) the amount of information they require about the Markov chain (e.g. the topology of the graph). In this paper, we provide an algorithm for SMC of unbounded properties, with confidence bounds, in the setting where only the minimum transition probability of the chain is known. Such an algorithm is particularly desirable in scenarios when the system is not known ("black box"), but also when it is too large to construct or fit into memory.

Most of the previous efforts in SMC have focused on the analysis of properties with bounded horizon [Younes and Simmons 2002; Sen et al. 2004; Younes et al. 2006; Younes and Simmons 2006; Jha et al. 2009; Jégourel et al. 2012; Bulychev et al. 2012]. For bounded properties (e.g. state r is reached with probability at most 0.5 in the first 1000 steps) statistical guarantees can be obtained in a completely black-box setting, where execution runs of the Markov chain can be observed, but no other information about the chain is available. Unbounded properties (e.g. state r is reached with probability at most 0.5 in any number of steps) are significantly more difficult, as a stopping criterion is needed when generating a potentially infinite execution run, and some information about the Markov chain is necessary for providing statistical guarantees (for an overview, see Table I). On the one hand, some approaches require the knowledge of the full topology in order to preprocess the Markov chain. On the other hand, when the topology is not accessible, there are approaches where the correctness of the statistics relies on information ranging from the second eigenvalue λ of the Markov chain, to knowledge of both the number $|S|$ of states and the minimum transition probability p_{min}.

Our contribution is a new SMC algorithm for full linear temporal logic (LTL), as well as for unbounded quantitative properties (mean payoff), which provides strong guarantees in the form of confidence bounds. Our algorithm uses less information about the Markov chain than previous algorithms that provide confidence bounds for unbounded properties—we need to know only the minimum transition probability p_{min} of the chain, and not the number of states nor the topology. Yet, experimentally, our algorithm performs in many cases better than these previous approaches (see Section 5). Our main idea is to *monitor each execution run on the fly in order to build statistical hypotheses about the structure of the Markov chain*. In particular, if from observing the current prefix of an execution run we can stipulate that with high probability a bottom strongly connected component (BSCC) of the chain has been entered and explored, then we can terminate the current execution run. The information obtained from execution prefixes allows us to terminate executions as soon as the property is decided with the required confidence, which is usually much earlier than any bounds that can be computed a priori. As far as we know, this is the first SMC algorithm that uses information obtained from execution prefixes.

Finding p_{min} is a light assumption in many realistic scenarios and often does not depend on the size of the chain – e.g. bounds on the rates for reaction kinetics in chemical reaction systems are typically known; alternatively, from a PRISM language model they can be easily inferred without constructing the respective state space.

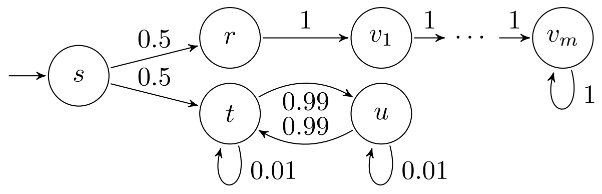


Fig. 1. A Markov chain.

Example 1.1. Consider the property of reaching state r in the Markov chain depicted in Figure 1. While the execution runs reaching r satisfy the property and can be stopped without ever entering any v_i, the finite execution paths without r, such as $stuttutuut$, are inconclusive. In other words, observing this path does not rule out the existence of a transition from, e.g., u to r, which, if existing, would eventually be taken with probability 1. This transition could have arbitrarily low probability, rendering its detection arbitrarily unlikely, yet its presence would change the probability of satisfying the property from 0.5 to 1. However, knowing that if there exists such a transition leaving the set, its transition probability is at least $p_{\mathsf{min}} = 0.01$, we can estimate the probability that the system is stuck in the set $\{t, u\}$ of states. Indeed, if existing, the exit transition was missed at least four times, no matter whether it exits t or u. Consequently, the probability that there is no such transition and $\{t, u\}$ is a BSCC is at least $1 - (1 - p_{\mathsf{min}})^4$.

This means that, in order to get 99% confidence that $\{t, u\}$ is a BSCC, we only need to see both t and u around 500 times[1] on a run. This is in stark contrast to a priori bounds that provide the same level of confidence, such as the $(1/p_{\mathsf{min}})^{|S|} = 100^{\mathcal{O}(m)}$ runs required by [Brázdil et al. 2014], which is infeasible for large m. In contrast, our method's performance is independent of m. $\triangle$

Monitoring execution prefixes allows us to design an SMC algorithm for complex unbounded properties such as full LTL. More precisely, we present a new SMC algorithm for LTL over Markov chains, specified as follows.

Input[2]**:**

— we can sample finite runs of arbitrary length from an unknown finite-state discrete-time Markov chain $\mathcal{M}$ according to the initial distribution[3],
— we are given a lower bound $p_{\mathsf{min}} > 0$ on the transition probabilities in $\mathcal{M}$,
— an LTL formula φ,
— a threshold probability p,
— an indifference region $\varepsilon > 0$,
— two error bounds $\alpha, \beta > 0$.

Output:

— if $\mathbb{P}[\varphi] \geq p + \varepsilon$, return YES with probability at least $1 - \alpha$, and
— if $\mathbb{P}[\varphi] \leq p - \varepsilon$, return NO with probability at least $1 - \beta$.

In addition, we present the first SMC algorithm for computing the mean payoff of Markov chains whose states are labelled with rewards.

[1] $1 - (1 - p_{\mathsf{min}})^{500} = 1 - 0.99^{500} \approx 0.993$
[2] Except for the transition probability bound p_{min}, all inputs are standard, as used in the literature, e.g. [Younes and Simmons 2002].
[3] We have a black-box system in the sense of [Sen et al. 2004], different from e.g. [Younes and Simmons 2002] or [Rabih and Pekergin 2009], where simulations can be run from any state.

Related work. Most of the effort in statistical model checking methods has focused on the analysis of properties with bounded horizon, e.g., [Younes and Simmons 2002; Sen et al. 2004; Younes et al. 2006; Younes and Simmons 2006; Jégourel et al. 2012; Bulychev et al. 2012]. These are properties whose satisfaction on a run can be decided based on its prefix of a fixed length. The unbounded properties are often investigated under the name "unbounded until" [He et al. 2010; Younes et al. 2010], which is only a slight generalisation of reachability.

SMC of unbounded properties was first considered in [Hérault et al. 2004]. It was suggested to try longer and longer simulations, but no bounds when to stop this process were given. The first solution was proposed in [Sen et al. 2005]. A simulation is to be stopped whenever we reach a point from which the goal state r cannot be reached at all. To this end, another set of simulations is run from such potential point to determine if there is any path to r. In order to avoid infinite simulations here, the simulations are stopped in each step with some "termination probability" p_{term}. This transforms the hypothesis testing task to one where simulations are almost surely finite. It was observed in [Younes and Simmons 2006] that this transformation works only on Markov chains that do not contain loops.

In [Lassaigne and Peyronnet 2008] the probability of unbounded property is approximated by a bounded variant that is sufficiently long. The correctness of this approach requires the second eigenvalue to be computed, which is as hard as the verification problem itself. A completely different approach is taken in [Rabih and Pekergin 2009]. Using coupling methods one can estimate the stationary distribution. However, the method is limited to ergodic Markov chains. In such a case all states of the system will be reached almost surely (and infinitely often).

Notably, in [Younes et al. 2010] two approaches are described. The first approach proposes to terminate sampled paths at every step with some probability p_{term}. In order to guarantee the asymptotic convergence of this method the second eigenvalue λ of the chain must be known, similar to [Lassaigne and Peyronnet 2008]. It should be noted that their method provides only asymptotic guarantees as the width of the confidence interval converges to zero. The second approach of [Younes et al. 2010] requires the knowledge of the chain's topology, which is used to transform the chain so that all potentially infinite paths are eliminated.

In [He et al. 2010] another transformation is performed, again requiring knowledge of the topology. This transformation assigns equal probability to all transitions leaving from a state, which effectively reduces checking of an unbounded until to a bounded variant. This method can only be used to check whether a property holds with a positive probability, but does not allow one to estimate the probability.

The (pre)processing of the state space required by the topology-aware methods, as well as by traditional numerical methods for Markov chain analysis, is a major practical hurdle for large (or unknown) state spaces. In [Brázdil et al. 2014] a priori bounds for the length of execution runs are calculated from the minimum transition probability and the number of states. However, without taking execution information into account, these bounds are exponential in the number of states and highly impractical, as illustrated in the example above.

There are also extensions of SMC to timed systems [David et al. 2015]. Our approach is also related to [Grosu and Smolka 2005; Oudinet et al. 2011], where the product of a non-deterministic system and Büchi automaton is explored for accepting lassos. We are not aware of any method for detecting BSCCs by observing a single run, employing no directed search of the state space.

To the best of our knowledge, we present the first SMC algorithm that provides confidence bounds for unbounded qualitative properties with access to only the minimum probability of the chain p_{min}, and the first SMC algorithm for quantitative properties.

Experimental evaluation. Our idea of inferring the structure of the Markov chain on the fly, while generating execution runs, allows for their early termination. In Section 5 we will see that for many chains arising in practice, such as the concurrent probabilistic protocols from the PRISM benchmark suite [Kwiatkowska et al. 2012], the BSCCs are reached quickly and, even more importantly, can be small even for very large systems. Consequently, many execution runs can be stopped quickly. Moreover, the number of execution runs necessary for a required precision and confidence is independent of the size of the state space, therefore this number can be small even for highly confident results (a good analogy is that of the opinion polls: the precision and confidence of opinion polls is regulated by the sample size and is independent of the size of the population). It is therefore not surprising that, experimentally, in most cases from the benchmark suite, our method outperforms previous methods (often even the numerical methods) despite requiring much less knowledge of the Markov chain, and despite providing strong guarantees in the form of confidence bounds. In Section 6, we also provide theoretical bounds on the running time of our algorithm for classes of Markov chains on which it performs particularly well.

Outline. The paper is organised as follows. Preliminaries are in Section 2. In Section 3 we describe our SMC method for unbounded reachability, and Section 4 presents extensions to linear temporal logic and mean payoff. Section 5 describes experimental evaluation of our method. In Section 6 we give a theoretical bound on the expected running time of our algorithms, and in Section 7 we present conclusions.

2. PRELIMINARIES

2.1. Markov chains

Definition 2.1 (*Markov chain*). A *Markov chain (MC)* is a tuple $\mathcal{M} = (S, \mathbf{P}, \mu)$, where

— S is a finite set of states,
— $\mathbf{P} : S \times S \to [0,1]$ is the transition probability matrix, such that for every $s \in S$ it holds $\sum_{s' \in S} \mathbf{P}(s, s') = 1$,
— μ is a probability distribution over S.

We let $p_{\mathsf{min}} := \min(\{\mathbf{P}(s,s') > 0 \mid s, s' \in S\})$ denote the smallest positive transition probability in $\mathcal{M}$. A *run* of $\mathcal{M}$ is an infinite sequence $\rho = s_0 s_1 \cdots$ of states, such that for all $i \geq 0$, $\mathbf{P}(s_i, s_{i+1}) > 0$; we let $\rho[i]$ denote the state s_i. A *path* π in $\mathcal{M}$ is a finite prefix of a run of $\mathcal{M}$. We denote the empty path by λ and concatenation of paths π_1 and π_2 by $\pi_1 \cdot \pi_2$. Each path π in $\mathcal{M}$ determines the set of runs $\mathsf{Cone}(\pi)$ consisting of all runs that start with π. To $\mathcal{M}$ we assign the probability space $(\mathsf{Runs}, \mathcal{F}, \mathbb{P}_{\mathcal{M}})$, where Runs is the set of all runs in $\mathcal{M}$, $\mathcal{F}$ is the σ-algebra generated by all $\mathsf{Cone}(\pi)$, and $\mathbb{P}_{\mathcal{M}}$ is the unique probability measure such that

$$\mathbb{P}_{\mathcal{M}}[\mathsf{Cone}(s_0 s_1 \cdots s_k)] = \mu(s_0) \cdot \prod_{i=1}^{k} \mathbf{P}(s_{i-1}, s_i),$$

where the empty product equals 1. We write $\mathbb{P}$ instead of $\mathbb{P}_{\mathcal{M}}$ if the Markov chain is clear from the context. The elements of $\mathcal{F}$ are called *events*. The respective expected value of a random variable $f : \mathsf{Runs} \to \mathbb{R}$ is $\mathbb{E}[f] = \int_{\mathsf{Runs}} f \, d\mathbb{P}$.

A non-empty set $C \subseteq S$ of states is *strongly connected* (SC) if for every $s, s' \in C$ there is a path from s to s'. A set of states $C \subseteq S$ is a *bottom strongly connected component* (BSCC) of $\mathcal{M}$, if it is a maximal SC, and for each $s \in C$ and $s' \in S \setminus C$ we have $\mathbf{P}(s, s') = 0$. The sets of all SCs and BSCCs in $\mathcal{M}$ are denoted by SC and BSCC, respectively. Note that with probability 1, the set of states that appear infinitely many times on a run forms a BSCC. From now on, we use the standard notions of SC and BSCC for directed graphs as well.

2.2. Hypothesis Testing

Let X be a random variable, and suppose we are interested whether the expected value $\mathbb{E}[X]$ is larger or smaller than some threshold p. We formulate this question as a hypothesis testing problem, where we decide between the *null hypothesis* H_0 and the *alternative hypothesis* H_1:

$$H_0 : \mathbb{E}[X] \geq p + \varepsilon \qquad\qquad H_1 : \mathbb{E}[X] < p - \varepsilon. \tag{1}$$

The indifference region $\varepsilon \geq 0$ describes the interval $[p - \varepsilon, p + \varepsilon)$ were both hypothesis are acceptable.

Two types of errors are used to evaluate precision of a solution. A *type I error* is the probability of accepting H_1 when H_0 holds. Similarly, a *type II error* is the probability of choosing H_0 when H_1 holds. The *test strength* (α, β) is a pair of values that bound the maximum probabilities of making type I and type II errors, respectively. In general, it is not possible to obtain low values of α and β at the same time when the indifference region ε is zero, since the probability $\mathbb{E}[X]$ may be arbitrary close to the threshold from either side, making type I or II error very likely.

Sequential probability ratio test. The *sequential probability ratio test* (SPRT) is a popular statistical procedure for hypothesis testing [Wald 1945; Younes 2004]. In the SPRT the number of samples is not fixed, but sampling continues until the observations give strong evidence in favor of H_0 or H_1. The SPRT gives no guarantee on the maximal number of samples; in practice, however, it often terminates quickly.

The SPRT works as follows. Suppose X is Bernoulli random variable, i.e. only values 0 and 1 are possible. After observing samples $\mathbf{x} = x_1, \ldots, x_n$ from X the following ratio is computed:

$$\frac{\mathbb{P}(\mathbf{x}|p_1)}{\mathbb{P}(\mathbf{x}|p_0)} = \prod_{i=1}^{n} \frac{\mathbb{P}(X = x_i \mid \mathbb{E}[X] = p_1)}{\mathbb{P}(X = x_i \mid \mathbb{E}[X] = p_0)} = \frac{p_1^{d_n}(1-p_1)^{n-d_n}}{p_1^{d_n}(1-p_0)^{n-d_n}},$$

where $d_n = \sum_{i=1}^{n} x_i$, $p_0 = p + \varepsilon$, and $p_1 = p - \varepsilon$. The decision rule for accepting a hypothesis is:

$$\text{accept } H_0 \text{ if } \frac{\mathbb{P}(\mathbf{x}|p_1)}{\mathbb{P}(\mathbf{x}|p_0)} \leq B \qquad \text{accept } H_1 \text{ if } \frac{\mathbb{P}(\mathbf{x}|p_1)}{\mathbb{P}(\mathbf{x}|p_0)} \geq A. \tag{2}$$

Finding the values of A, B such the test has the required strength is a difficult task. In practice, values $A = (1-\beta)/\alpha$ and $B = \beta/(1-\alpha)$ are used, since they result in a test whose strength is close to (α, β) [Younes 2004].

3. SOLUTION FOR REACHABILITY

A fundamental problem in Markov chain verification is computing the probability that a certain set of goal states is reached. For the rest of the paper, let $\mathcal{M} = (S, \mathbf{P}, \mu)$ be a Markov chain and $G \subseteq S$ be the set of the goal states in $\mathcal{M}$. We let

$$\Diamond G := \{\rho \in \mathsf{Runs} \mid \exists i \geq 0 : \rho[i] \in G\}$$

denote the event that "eventually a state in G is reached." The event $\Diamond G$ is measurable and its probability $\mathbb{P}[\Diamond G]$ can be computed numerically or estimated using statistical algorithms. Since no bound on the number of steps for reaching G is given, the major difficulty for any statistical approach is to decide how long each sampled path should be. We can stop extending the path either when we reach G, or when no more new states can be reached anyways. The latter happens if and only if we are in a BSCC and we have seen all of its states.

In this section, we first show how to monitor each simulation run on the fly, in order to detect quickly if a BSCC has been entered with high probability. Then, we show how to use hypothesis testing in order to estimate $\mathbb{P}[\Diamond G]$.

3.1. BSCC detection

We start with an example illustrating how to measure probability of reaching a BSCC from one path observation.

Example 3.1. Recall Example 1 and Figure 1. Now, consider an execution path $stuttutu$. Intuitively, does $\{t, u\}$ look as a good "candidate" for being a BSCC of $\mathcal{M}$? We visited both t and u three times; we have taken a transition from each t and u at least twice without leaving $\{t, u\}$. By the same reasoning as in Example 1, we could have missed some outgoing transition with probability at most $(1 - p_{\mathsf{min}})^2$. The structure of the system that can be deduced from this path is in Figure 2 and is correct with probability at least $1 - (1 - p_{\mathsf{min}})^2$. △

Now we formalise our intuition. Given a finite or infinite sequence $\rho = s_0 s_1 \cdots$, the *support* of ρ is the set $\overline{\rho} = \{s_0, s_1, \ldots\}$. Further, the *graph of* ρ is given by vertices $\overline{\rho}$ and edges $\{(s_i, s_{i+1}) \mid i = 0, 1, \ldots\}$.

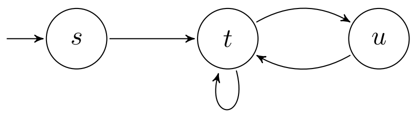


Fig. 2. The graph of a path $stuttutu$.

Definition 3.2 (*Candidate*). If a path π has a suffix κ such that $\overline{\kappa}$ is a BSCC of the graph of π, we call $\overline{\kappa}$ the *candidate of* π. Moreover, for $k \in \mathbb{N}$, we call it a *k-candidate (of π)* if each $s \in \overline{\kappa}$ has at least k occurrences in κ and the last element of κ has at least $k+1$ occurrences. A *k-candidate of a run* ρ is a k-candidate of some prefix of ρ.

Note that for each path there is at most one candidate. Therefore, we write $K(\pi)$ to denote the candidate of π if there is one, and $K(\pi) = \bot$, otherwise. Observe that each $K(\pi) \neq \bot$ is strongly connected in $\mathcal{M}$.

Example 3.3. Consider a path $\pi = stuttutu$, then $K(\pi) = \{t, u\}$. Observe that $\{t\}$ is not a candidate as it is not maximal. Further, $K(\pi)$ is a 2-candidate (and as such also a 1-candidate), but not a 3-candidate. Intuitively, the reason is that we only took a transition from u (to the candidate) twice, cf. Example 3.1. △

Intuitively, the higher the k the more it looks as if the k-candidate is indeed a BSCC. Denoting by $Cand_k(K)$ the random predicate of K being a k-candidate on a run, the probability of "unluckily" detecting any specific non-BSCC set of states K as a k-candidate, can be bounded as follows.

LEMMA 3.4. *For every $K \subseteq S$ such that $K \notin \mathsf{BSCC}$, and every $s \in K$, $k \in \mathbb{N}$,*

$$\mathbb{P}[Cand_k(K) \mid \Diamond s] \leq (1 - p_{\mathsf{min}})^k .$$

PROOF. Since K is not a BSCC, there is a state $t \in K$ with a transition to $t' \notin K$. The set of states K is a k-candidate of a run, only if t is visited at least k times by the path and was never followed by t' (indeed, even if t is the last state in the path, by definition of a k-candidate, there are also at least k previous occurrences of t in the path). Further, since the transition from t to t' has probability at least p_{min}, the probability of not taking the transition k times is at most $(1 - p_{\mathsf{min}})^k$.

□

Example 3.5. We illustrate how candidates "evolve over time" along a run. Consider a run $\rho = s_0 s_0 s_1 s_0 \cdots$ of the Markov chain in Figure 3. The empty and one-letter prefix do not have the candidate defined, $s_0 s_0$ has a candidate $\{s_0\}$, then again $K(s_0 s_0 s_1) = \bot$, and $K(s_0 s_0 s_1 s_0) = \{s_0, s_1\}$. One can observe that subsequent candidates are either disjoint or contain some of the previous candidates. Consequently, there are at most $2|S| - 1$ candidates on every run, which is in our setting an unknown bound. △

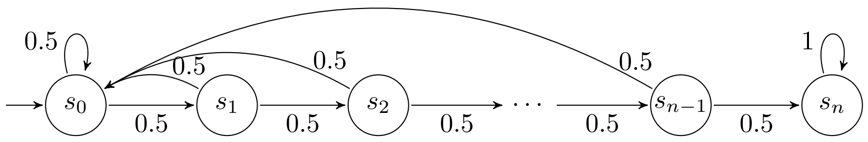


Fig. 3. A family (for $n \in \mathbb{N}$) of Markov chains with large eigenvalues.

While we have bounded the probability of detecting any specific non-BSCC set K as a k-candidate, we need to bound the overall error for detecting a candidate that is not a BSCC. Since there can be many false candidates on a run before the real BSCC (e.g. Figure 3), we need to bound the error of reporting any of them.

In the following, we first formalise the process of discovering candidates along the run. Second, we bound the error that any of the non-BSCC candidates becomes a k-candidate. Third, we bound the overall error of not detecting the real BSCC by increasing k every time a different candidate is found.

We start with discovering the sequence of candidates on a run. For a run $\rho = s_0 s_1 \cdots$, consider the sequence of random variables defined by $K(s_0 \ldots s_j)$ for $j \geq 0$, and let $(K_i)_{i \geq 1}$ be the subsequence without undefined elements and with no repetition of consecutive elements. For example, for a run $\rho = s_0 s_1 s_1 s_1 s_0 s_1 s_2 s_2 \cdots$, we have $K_1 = \{s_1\}$, $K_2 = \{s_0, s_1\}$, $K_3 = \{s_2\}$, etc. Let K_j be the last element of this sequence, called the *final candidate*. Additionally, we define $K_\ell := K_j$ for all $\ell > j$. We describe the lifetime of a candidate. Given a non-final K_i, we write $\rho = \alpha_i \beta_i b_i \gamma_i d_i \delta_i$ so that $\overline{\alpha_i} \cap K_i = \emptyset$, $\overline{\beta_i b_i \gamma_i} = K_i$, $d_i \notin K_i$, and $K(\alpha_i \beta_i) \neq K_i$, $K(\alpha_i \beta_i b_i) = K_i$. Intuitively, we start exploring K_i in β_i; K_i becomes a candidate in b_i, the birthday of the ith candidate; it remains to be a candidate until d_i, the death of the ith candidate. For example, for the run $\rho = s_0 s_1 s_1 s_1 s_0 s_1 s_2 s_2 \cdots$ and $i = 1$, $\alpha_1 = s_0$, $\beta_1 = s_1$, $b_1 = s_1$, $\gamma_1 = s_1$, $d_1 = s_0$, $\delta_1 = s_1 s_2 s_2 \rho[8] \rho[9] \cdots$. Note that the final candidate is almost surely a BSCC of $\mathcal{M}$ and would thus have γ_j infinite.

Now, we proceed to bounding errors for each candidate. Since there is an unknown number of candidates on a run, we will need a slightly stronger definition. First, observe that $Cand_k(K_i)$ iff K_i is a k-candidate of $\beta_i b_i \gamma_i$. We say K_i is a *strong k-candidate*, written $SCand_k(K_i)$, if it is a k-candidate of $b_i \gamma_i$. Intuitively, it becomes a k-candidate even not counting the discovery phase. As a result, even if we already assume there exists an ith candidate, its strong k-candidacy gives the guarantees on being a BSCC as above in Lemma 3.4.

LEMMA 3.6. *For every $i, k \in \mathbb{N}$, we have*

$$\mathbb{P}[SCand_k(K_i) \mid K_i \notin \mathsf{BSCC}] \leq (1 - p_{\mathsf{min}})^k \,.$$

PROOF.

$$\begin{aligned}
&\mathbb{P}[SCand_k(K_i) \mid K_i \notin \mathsf{BSCC}] \\
&= \frac{1}{\mathbb{P}[K_i \notin \mathsf{BSCC}]} \sum_{\substack{C \in \mathsf{SC} \setminus \mathsf{BSCC} \\ s \in C}} \mathbb{P}[K_i = C, b_i = s] \cdot \mathbb{P}[SCand_k(C) \mid K_i = C, b_i = s] \\
&= \frac{1}{\mathbb{P}[K_i \notin \mathsf{BSCC}]} \sum_{\substack{C \in \mathsf{SC} \setminus \mathsf{BSCC} \\ s \in C}} \mathbb{P}[K_i = C, b_i = s] \cdot \mathbb{P}[Cand_k(C) \mid \Diamond s] \quad \text{(by Markov property)} \\
&\leq \frac{1}{\mathbb{P}[K_i \notin \mathsf{BSCC}]} \sum_{\substack{C \in \mathsf{SC} \setminus \mathsf{BSCC} \\ s \in C}} \mathbb{P}[K_i = C, b_i = s] \cdot (1 - p_{\mathsf{min}})^k \quad \text{(by Lemma 3.4)} \\
&= (1 - p_{\mathsf{min}})^k \,. \quad \text{(by } \mathbb{P}[K_i \in \mathsf{SC}, b_i \in K_i] = 1\text{)}
\end{aligned}$$

□

Since the number of candidates can only be bounded with some knowledge of the state space, e.g. its size, we assume no bounds and provide a method to bound the error even for an unbounded number of candidates on a run.

LEMMA 3.7. *For $(k_i)_{i=1}^{\infty} \in \mathbb{N}^{\mathbb{N}}$, let $\mathcal{E}rr$ be the set of runs such that for some $i \in \mathbb{N}$, we have $SCand_{k_i}(K_i)$ despite $K_i \notin \mathsf{BSCC}$. Then*

$$\mathbb{P}[\mathcal{E}rr] < \sum_{i=1}^{\infty} (1 - p_{\mathsf{min}})^{k_i} \,.$$

PROOF.

$$\begin{aligned}
\mathbb{P}[\mathcal{E}rr] &= \mathbb{P}\left[\bigcup_{i=1}^{\infty} \Big(SCand_{k_i}(K_i) \cap K_i \notin \mathsf{BSCC}\Big)\right] \\
&\leq \sum_{i=1}^{\infty} \mathbb{P}[SCand_{k_i}(K_i) \cap K_i \notin \mathsf{BSCC}] \quad \text{(by the union bound)} \\
&= \sum_{i=1}^{\infty} \mathbb{P}[SCand_{k_i}(K_i) \mid K_i \notin \mathsf{BSCC}] \cdot \mathbb{P}[K_i \notin \mathsf{BSCC}] \\
&\leq \sum_{i=1}^{\infty} \mathbb{P}[SCand_{k_i}(K_i) \mid K_i \notin \mathsf{BSCC}] \\
&= \sum_{i=1}^{\infty} (1 - p_{\mathsf{min}})^{k_i} \,. \quad \text{(by Lemma 3.6)}
\end{aligned}$$

□

In Algorithm 1 we present a procedure for deciding whether a BSCC inferred from a path π is indeed a BSCC with confidence greater than $1 - \delta$. We use notation $\mathrm{SC}\small{\mathrm{AND}}_{k_i}(K, \pi)$ to denote the function deciding whether K is a strong k_i-candidate on π. The overall error bound is obtained by setting $k_i = \frac{i - \log \delta}{-\log(1 - p_{\mathsf{min}})}$.

Algorithm 1 REACHEDBSCC

```
Input: path $\pi = s_0 s_1 \cdots s_n$, $p_{\mathsf{min}}, \delta \in (0,1]$
Output: Yes iff $K(\pi) \in \mathsf{BSCC}$
  $C \leftarrow \bot$, $i \leftarrow 0$
  for $j = 0$ to $n$ do
      if $K(s_0 \cdots s_j) \neq \bot$ and $K(s_0 \cdots s_j) \neq C$ then
          $C \leftarrow K(s_0 \cdots s_j)$
          $i \leftarrow i + 1$
  $k_i \leftarrow \frac{i - \log \delta}{-\log(1 - p_{\mathsf{min}})}$
  if $i \geq 1$ and SCAND$_{k_i}(K(\pi), \pi)$ then return Yes
  else return No
```

THEOREM 3.8. *For every $\delta > 0$, Algorithm 1 is correct with error probability at most δ.*

PROOF. Since M is finite, the Algorithm 1 terminates almost surely. The probability to return an incorrect result can be bounded by returning incorrect result for one of the non-final candidates, which by Lemma 3.7 is as follows:

$$\sum_{i=1}^{\infty}(1 - p_{\mathsf{min}})^{k_i} = \sum_{i=1}^{\infty}(1 - p_{\mathsf{min}})^{\frac{-i+\log \delta}{\log(1-p_{\mathsf{min}})}} = \sum_{i=1}^{\infty} 2^{-i+\log \delta} = \sum_{i=1}^{\infty} \delta/2^i = \delta\ .$$

□

We have shown how to detect a BSCC of a single path with desired confidence. In Algorithm 2, we show how to use our BSCC detection method to decide whether a given path reaches the set G with confidence $1 - \delta$. The function $\mathsf{NextState}(\pi)$ randomly picks a state according to the initial distribution μ if the path is empty ($\pi = \lambda$); otherwise, if ℓ is the last state of π, it randomly chooses its successor according to $\mathbf{P}(\ell, \cdot)$. The algorithm returns Yes when π reaches a state in G, and No when for some i, the ith candidate is a strong k_i-candidate. In the latter case, with probability at least $1 - \delta$, π has reached a BSCC not containing G. Hence, with probability at most δ, the algorithm returns No for a path that could reach a goal.

Algorithm 2 SINGLEPATHREACH

```
Input: goal states $G$ of $\mathcal{M}$, $p_{\mathsf{min}}, \delta \in (0,1]$
Output: Yes iff a run reaches $G$
  $\pi \leftarrow \lambda$
  repeat
      $s \leftarrow \mathsf{NextState}(\pi)$
      $\pi \leftarrow \pi \cdot s$
      if $s \in G$ then return Yes                 ▷ We have provably reached $G$
  until REACHEDBSCC($\pi, p_{\mathsf{min}}, \delta$)
  return No                  ▷ By Theorem 3.8, $\mathbb{P}[K(\pi) \in \mathsf{BSCC}] \geq 1 - \delta$
```

3.2. Hypothesis testing with bounded error

In the following, we show how to estimate the probability of reaching a set of goal states, by combining the BSCC detection and hypothesis testing. More specifically, we sample many paths of a Markov chain, decide for each whether it reaches the goal states (Algorithm 2), and then use hypothesis testing to estimate the event probability.

The hypothesis testing is adapted to the fact that testing reachability on a single path may report false negatives.

Let $X_\diamond^\delta$ be a Bernoulli random variable, such that $X_\diamond^\delta = 1$ if and only if SINGLEPATHREACH$(G, p_{\mathsf{min}}, \delta)$ = **Yes**, describing the outcome of Algorithm 2. The following theorem establishes that $X_\diamond^\delta$ estimates $\mathbb{P}[\diamond G]$ with a bias bounded by δ.

THEOREM 3.9. *For every $\delta > 0$, we have $\mathbb{P}[\diamond G] - \delta \leq \mathbb{E}[X_\diamond^\delta] \leq \mathbb{P}[\diamond G]$.*

PROOF. Since the event $\diamond G$ is necessary for $X_\diamond^\delta = 1$, we have $\mathbb{P}[\diamond G \mid X_\diamond^\delta = 1] = 1$. It follows that $\mathbb{E}[X_\diamond^\delta] = \mathbb{P}[X_\diamond^\delta = 1] = \mathbb{P}[\diamond G, X_\diamond^\delta = 1] \leq \mathbb{P}[\diamond G]$, hence the upper bound. As for the lower bound:

$$\begin{aligned}
\mathbb{E}[X_\diamond^\delta] = \mathbb{P}[X_\diamond^\delta = 1] &= \mathbb{P}[\diamond G, X_\diamond^\delta = 1] && \diamond G \text{ is necessary for } X_\diamond^\delta = 1\\
&= \mathbb{P}[\diamond G] - \mathbb{P}[\diamond G, X_\diamond^\delta = 0] \\
&\geq \mathbb{P}[\diamond G] - \delta\ . && \text{by Theorem 3.8}
\end{aligned}$$

□

In order to conclude on the value $\mathbb{P}[\diamond G]$, the standard statistical model checking approach via hypothesis testing (cf. Section 2.2) decides between the hypothesis

$$H_0 : \mathbb{P}(\diamond G) \geq p + \varepsilon \qquad\qquad H_1 : \mathbb{P}(\diamond G) < p - \varepsilon\ .$$

where ε is a desired indifference region. As we do not have precise observations on each path, we reduce this problem to a hypothesis testing on the variable $X_\diamond^\delta$ with a narrower indifference region:

$$H_0' : \mathbb{E}[X_\diamond^\delta] \geq p + (\varepsilon - \delta) \qquad\qquad H_1' : \mathbb{E}[X_\diamond^\delta] < p - \varepsilon,$$

for some $\delta < \varepsilon$.

We define the reduction simply as follows. Given a statistical test $\mathcal{T}'$ for H_0', H_1' we define a test $\mathcal{T}$ that accepts H_0 if $\mathcal{T}'$ accepts H_0', and H_1 otherwise. The following lemma shows that $\mathcal{T}$ has the same strength as $\mathcal{T}'$.

LEMMA 3.10. *Suppose the test $\mathcal{T}'$ decides between H_0' and H_1' with strength (α, β). Then the test $\mathcal{T}$ decides between H_0 with H_1 with strength (α, β).*

PROOF. Consider type I error of $\mathcal{T}$. Assume that H_0 holds, which means $\mathbb{P}[\diamond G] \geq p + \varepsilon$. By Theorem 3.9 it follows that $\mathbb{P}[X_\diamond^\delta = 1] \geq \mathbb{P}[\diamond G] - \delta \geq p + (\varepsilon - \delta)$, thus H_0' also holds. By assumption the test $\mathcal{T}'$ accepts H_1' with probability at most α, thus, by the reduction, $\mathcal{T}$ also accepts H_1 with probability $\leq \alpha$. The proof for type II error is analogous.

□

Lemma 3.10 gives us the following algorithm to decide between H_0 and H_1. We generate samples $x_0, x_1, \cdots, x_n \sim X_\diamond^\delta$ from SINGLEPATHREACH$(G, p_{\mathsf{min}}, \delta)$, and apply a statistical test to decide between H_0' and H_1'. Finally, we accept H_0 if H_0' was accepted by the test, and H_1 otherwise.

4. EXTENSIONS

In this section, we present how the on-the-fly BSCC detection can be used for verifying LTL and quantitative properties (mean payoff).

4.1. Linear temporal logic

We show how our method extends to properties expressible by linear temporal logic (LTL) [Pnueli 1977] and, in the same manner, to all ω-regular properties. Given a finite set Ap of atomic propositions, a *labelled Markov chain* (LMC) is a tuple $\mathcal{M} = (S, \mathbf{P}, \mu, L)$, where $(S, \mathbf{P}, \mu)$ is a MC and $L : S \to 2^{Ap}$ is a labelling function. The formulae of LTL are given by the following syntax:

$$\varphi \ ::= \ a \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathbf{X}\varphi \mid \varphi \mathbf{U} \varphi$$

for $a \in Ap$. The semantics is defined with respect to a word $w \in (2^{Ap})^{\omega}$. The ith letter of w is denoted by $w[i]$, i.e. $w = w[0]w[1]\cdots$ and we write w_i for the suffix $w[i]w[i+1]\cdots$. We define

$$\begin{array}{lll} w \models a & \iff & a \in w[0] \\ w \models \neg\varphi & \iff & \text{not } w \models \varphi \\ w \models \varphi \wedge \psi & \iff & w \models \varphi \text{ and } w \models \psi \\ w \models \mathbf{X}\varphi & \iff & w_1 \models \varphi \\ w \models \varphi \mathbf{U} \psi & \iff & \exists\, k \in \mathbb{N} : (w_k \models \psi \text{ and } \forall\, 0 \le j < k : w_j \models \varphi\ ). \end{array}$$

The set $\{w \in (2^{Ap})^{\omega} \mid w \models \varphi\}$ is denoted by $\mathsf{L}(\varphi)$. Given a labelled Markov chain $\mathcal{M}$ and an LTL formula φ, we are interested in the measure

$$\mathbb{P}_{\mathcal{M}}[\varphi] := \mathbb{P}_{\mathcal{M}}[\{\rho \in \mathsf{Runs} \mid L(\rho) \models \varphi\}],$$

where L is naturally extended to runs by $L(\rho)[i] = L(\rho[i])$ for all i.

For every LTL formula φ, one can construct a deterministic Rabin automaton that accepts all runs that satisfy φ.

Definition 4.1 (*Deterministic Rabin automaton*). A *deterministic Rabin automaton* (DRA) is a tuple $\mathcal{A} = (Q, 2^{Ap}, \gamma, q_o, Acc)$, where

— Q is a finite set of states,
— $\gamma : Q \times 2^{Ap} \to Q$ is the transition function,
— $q_o \in Q$ is the initial state, and
— $Acc \subseteq 2^Q \times 2^Q$ is the acceptance condition.

A word $w \in (2^{Ap})^{\omega}$ induces an infinite sequence $\mathcal{A}(w) = s_0 s_1 \cdots \in Q^{\omega}$, such that $s_0 = q_0$ and $\gamma(s_i, w[i]) = s_{i+1}$ for $i \ge 0$. We write $\mathrm{Inf}(w)$ for the set of states that occur infinitely often in $\mathcal{A}(w)$. Word w is accepted, if there exists a pair $(E, F) \in Acc$, such that $E \cap \mathrm{Inf}(w) = \emptyset$ and $F \cap \mathrm{Inf}(w) \neq \emptyset$. The language $\mathsf{L}(\mathcal{A})$ of $\mathcal{A}$ is the set of all words accepted by $\mathcal{A}$. The following is a well known result, see e.g. [Baier and Katoen 2008].

LEMMA 4.2. *For every LTL formula φ, a DRA $\mathcal{A}_{\varphi}$ can be effectively constructed such that $\mathsf{L}(\mathcal{A}_{\varphi}) = \mathsf{L}(\varphi)$.*

The product of a MC and DRA is defined in the following way.

Definition 4.3 (*Product of a MC and DRA*). The product of a Markov chain $\mathcal{M} = (S, \mathbf{P}, \mu)$ and deterministic Rabin automaton $\mathcal{A} = (Q, 2^{Ap}, \gamma, q_o, Acc)$ is the Markov chain $\mathcal{M} \otimes \mathcal{A} = (S \times Q, \mathbf{P}', \mu')$, where

— $\mathbf{P}'((s, q), (s', q')) = \mathbf{P}(s, s')$ if $q' = \gamma(q, L(s'))$ and $\mathbf{P}'((s, q), (s', q')) = 0$ otherwise,
— $\mu'(s, q) = \mu(s)$ if $\gamma(q_o, L(s)) = q$ and $\mu'(s, q) = 0$ otherwise.

Note that $\mathcal{M} \otimes \mathcal{A}$ has the same smallest transition probability p_{min} as $\mathcal{M}$.

The crux of LTL probabilistic model checking relies on the fact that the probability of satisfying an LTL property φ in a Markov chain $\mathcal{M}$ equals the probability of reaching an accepting BSCC in the Markov chain $\mathcal{M} \otimes \mathcal{A}_{\varphi}$. Formally, a BSCC C of $\mathcal{M} \otimes \mathcal{A}_{\varphi}$ is *accepting* if for some $(E, F) \in Acc$ we have $C \cap (S \times E) = \emptyset$ and $C \cap (S \times F) \neq \emptyset$. Let $\mathsf{AccBSCC}$ denote the union of all accepting BSCCs in $\mathcal{M} \otimes \mathcal{A}_{\varphi}$. Then we obtain the following well-known fact [Baier and Katoen 2008]:

LEMMA 4.4. *For every labelled Markov chain $\mathcal{M}$ and LTL formula φ, we have* $\mathbb{P}_{\mathcal{M}}[\varphi] = \mathbb{P}_{\mathcal{M}\otimes\mathcal{A}_\varphi}[\Diamond \mathsf{AccBSCC}]$.

Algorithm 3 SINGLEPATHLTL

```
Input: DRA $\mathcal{A} = (Q, 2^{Ap}, \gamma, q_o, Acc)$, $p_{\min}$, $\delta \in (0,1]$
Output: Yes iff the final candidate is an accepting BSCC
  $q \leftarrow q_o$, $\pi \leftarrow \lambda$
  repeat
      $s \leftarrow \mathsf{NextState}(\pi)$
      $q \leftarrow \gamma(q, L(s))$
      $\pi \leftarrow \pi \cdot (s, q)$
  until REACHEDBSCC($\pi, p_{\min}, \delta$)                ▷ $\mathbb{P}[K(\pi) \in \mathsf{BSCC}] \geq 1 - \delta$
  return $\exists (E,F) \in Acc : K(\pi) \cap (S \times E) = \emptyset \wedge K(\pi) \cap (S \times F) \neq \emptyset$
```

4.2. Hypothesis testing with bounded error

Since the input used is a Rabin automaton, the method applies to all ω-regular properties. Let X_φ^δ be a Bernoulli random variable, such that $X_\varphi^\delta = 1$ if and only if SINGLEPATHLTL$(\mathcal{A}_\varphi, p_{\min}, \delta) =$ **Yes**. Since the BSCC must be reached and fully explored to classify it correctly, the error of the algorithm can now be both-sided.

THEOREM 4.5. *For every $\delta > 0$, $\mathbb{P}[\varphi] - \delta \leq \mathbb{E}[X_\varphi^\delta] \leq \mathbb{P}[\varphi] + \delta$.*

Further, like in Section 3.2, we can reduce the hypothesis testing problem for

$$H_0 : \mathbb{P}[\varphi] \geq p + \varepsilon \qquad \text{and} \qquad H_1 : \mathbb{P}[\varphi] \leq p - \varepsilon$$

for any $\delta < \varepsilon$ to the following hypothesis testing problem on the observable X_φ^δ

$$H_0' : \mathbb{E}[X_\varphi^\delta] \geq p + (\varepsilon - \delta) \qquad \text{and} \qquad H_1' : \mathbb{E}[X_\varphi^\delta] \leq p - (\varepsilon - \delta)\,.$$

4.3. Mean payoff

We show that our method extends also to quantitative properties, such as *mean payoff* (also called long-run average reward). Let $\mathcal{M} = (S, \mathbf{P}, \mu)$ be a Markov chain and $r : S \to [0,1]$ be a *reward* function. Denoting by S_i the random variable returning the i-th state on a run, the aim is to compute

$$\mathsf{MP} := \lim_{n\to\infty} \mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n} r(S_i)\right]\,.$$

This limit exists (see, e.g. [Norris 1998]), and equals $\sum_{C\in\mathsf{BSCC}} \mathbb{P}[\Diamond C] \cdot \mathsf{MP}_C$,where MP_C is the mean payoff of runs ending in C. Note that MP_C can be computed from r and transition probabilities in C [Norris 1998]. We have already shown how our method estimates $\mathbb{P}[\Diamond C]$. Now we show how it extends to estimating transition probabilities in BSCCs and thus the mean payoff.

First, we focus on a single path π that has reached a BSCC $C = K(\pi)$ and show how to estimate the transition probabilities $\mathbf{P}(s, s')$ for each $s, s' \in C$. Let $X_{s,s'}$ be the random variable denoting the event that $\mathsf{NextState}(s) = s'$. $X_{s,s'}$ is a Bernoulli variable with parameter $\mathbf{P}(s, s')$, so we use the obvious estimator $\hat{\mathbf{P}}(s, s') = \#_{ss'}(\pi)/\#_s(\pi)$, where $\#_\alpha(\pi)$ is the number of occurrences of α in π. If π is long enough so that $\#_s(\pi)$

is large enough, the estimation is guaranteed to have desired precision ξ with desired confidence $(1-\delta_{s,s'})$. Indeed, using Höffding's inequality, we obtain

$$\mathbb{P}[\hat{\mathbf{P}}(s,s') - \mathbf{P}(s,s')| > \xi] \le \delta_{s,s'} = 2e^{-2\#_s(\pi)\cdot\xi^2} \, . \tag{3}$$

Hence, we can extend the path π with candidate C until it is long enough so that we have a $1-\delta_C$ confidence that all the transition probabilities in C are in the ξ-neighbourhood of our estimates, by ensuring that $\sum_{s,s'\in C}\delta_{s,s'} < \delta_C$. These estimated transition probabilities $\hat{\mathbf{P}}$ induce an *estimated mean payoff* $\hat{\mathsf{MP}}_C$. The following theorem relates the estimated and exact mean payoff.

THEOREM 4.6. *Let C be a BSCC in a Markov chain $\mathcal{M}$ with rewards in the range $[0,1]$, MP_C be the mean payoff of C, and $\hat{\mathsf{MP}}_C$ be the estimated mean payoff of C. Then*

$$|\hat{\mathsf{MP}}_C - \mathsf{MP}_C| \le \zeta := \left(1 + \frac{\xi}{p_{\mathsf{min}}}\right)^{2\cdot|C|} - 1 \, . \tag{4}$$

PROOF. Consider a Markov chain C with a reward function $r : S \to [0,1]$, such that C is a single BSCC. The discounted sum MD^λ for a state s of C is defined as:

$$\mathsf{MD}^\lambda(s) := \lim_{n\to\infty} \mathbb{E}\left[\frac{\sum_{i=1}^n r(S_i)\lambda^i}{\sum_{i=1}^n \lambda^i}\right],$$

where $\lambda > 0$ is a discount factor. We say that a Markov chain $\hat{C}$ is ξ-close to C if

(1) $\hat{C}$ is over the same states as C,
(2) $\forall s,s' \in C : |\mathbf{P}_C(s,s') - \mathbf{P}_{\hat{C}}(s,s')| \le \xi$,
(3) $\forall s,s' \in C : \mathbf{P}_C(s,s') > 0 \iff \mathbf{P}_{\hat{C}}(s,s') > 0$.

We write $\hat{\mathsf{MD}}^\lambda$ for the discounted sum computed for $\hat{C}$. By [Chatterjee 2012](Theorem 4) it holds that for every discount factor $0 < \lambda < 1$, every MC $\hat{C}$ that is ξ-close to C, and every state s:

$$|\hat{\mathsf{MD}}^\lambda(s) - \mathsf{MD}^\lambda(s)| \le \left(1 + \frac{\xi}{p_{\mathsf{min}}}\right)^{2\cdot|C|} - 1 \, , \tag{5}$$

where p_{min} is the minimum transition probability in $\mathcal{M}$. By [Solan 2003] we know that the discounted sum converges to mean payoff:

$$\lim_{\lambda\to 1} \mathsf{MD}^\lambda(s) = \mathsf{MP}_C \qquad\qquad \lim_{\lambda\to 1} \hat{\mathsf{MD}}^\lambda(s) = \hat{\mathsf{MP}}_C,$$

where MP_C and $\hat{\mathsf{MP}}_C$ are the mean payoff for C and $\hat{C}$, respectively. We obtain the result by taking the limit $\lambda \to 1$ in (5).

□

Note that by Taylor's expansion, for small ξ, we have $\zeta \approx 2|C|\xi$.

Algorithm 4 extends Algorithm 2 as follows. It divides the confidence parameters δ into δ_{BSCC} (used as in Algorithm 2 to detect the BSCC) and δ_C (the total confidence for the estimates on transition probabilities). For simplicity, we set $\delta_{BSCC} = \delta_C = \delta/2$. First, we compute the bound ξ required for ζ-precision (by Eq. 4). Subsequently, we compute the required strength k of the candidate guaranteeing δ_C-confidence on $\hat{\mathbf{P}}$ (from Eq. 3). The path is prolonged until the candidate is strong enough; in such a case $\hat{\mathsf{MP}}_C$ is ζ-approximated with $1-\delta_C$ confidence. If the candidate of the path changes, all values are computed from scratch for the new candidate.

Algorithm 4 SINGLEPATHMP

```
Input: reward function $r$, $p_{\mathsf{min}}, \zeta, \delta \in (0,1]$,
Output: $\hat{\mathsf{MP}}_C$ such that $|\hat{\mathsf{MP}}_C - \mathsf{MP}_C| < \zeta$ where $C$ is the BSCC of the generated run
  $\pi \leftarrow \lambda$
  repeat
      $\pi \leftarrow \pi\,.\,\mathsf{NextState}(\pi)$
      if $K(\pi) \neq \bot$ then
          $\xi = p_{\mathsf{min}}((1+\zeta)^{1/2|K(\pi)|} - 1)$                    ▷ By Equation (4)
          $k \leftarrow \frac{\ln(2|K(\pi)|^2) - \ln(\delta/2)}{2\xi^2}$                    ▷ By Equation (3)
  until REACHEDBSCC($\pi, p_{\mathsf{min}}, \delta/2$) and SCAND$_k(K(\pi), \pi)$
  return $\hat{\mathsf{MP}}_{K(\pi)}$ computed from $\hat{\mathbf{P}}$ and $r$
```

THEOREM 4.7. *For every $\delta > 0$, the Algorithm 4 terminates correctly with probability at least $1 - \delta$.*

PROOF. From Eq. 3, by the union bound, we are guaranteed that the probability that *none* of the estimates $\hat{\mathbf{P}}_{s,s'}$ is outside of the ζ-neighbourhood doesn't exceed the sum of all respective estimation errors, that is, $\delta_C = \sum_{s,s' \in C} \delta_{s,s'}$. Next, from Eq. 4 and from the fact that C is subject to Theorem 3.8 with confidence δ_{BSCC},

$$\begin{aligned}
P(|\mathsf{MP}_C(r) - \hat{\mathsf{MP}}_C(r)| > \zeta) = \\
&= P(C \in \mathsf{BSCC}) P(|\mathsf{MP}(r) - \hat{\mathsf{MP}}(r)| > \zeta \mid C \in \mathsf{BSCC}) + \\
&\quad P(C \notin \mathsf{BSCC}) P(|\mathsf{MP}(r) - \hat{\mathsf{MP}}(r)| > \zeta \mid C \notin \mathsf{BSCC}) \\
&\leq 1 \cdot \delta_C + \delta_{BSCC} \cdot 1 = \delta_C + \delta_{BSCC} \leq \delta.
\end{aligned}$$

□

4.4. Hypothesis testing with bounded error

Let random variable $X^{\zeta,\delta}_{\mathsf{MP}}$ denote the value SINGLEPATHMP($r, p_{\mathsf{min}}, \zeta, \delta$). The following theorem establishes relation between the mean-payoff MP and the expected value of $X^{\zeta,\delta}_{\mathsf{MP}}$.

THEOREM 4.8. *For every $\delta, \zeta > 0$, $\mathsf{MP} - \zeta - \delta \leq \mathbb{E}[X^{\zeta,\delta}_{\mathsf{MP}}] \leq \mathsf{MP} + \zeta + \delta$.*

PROOF. Let us write $X^{\zeta,\delta}_{\mathsf{MP}}$ as an expression of random variables Y, W, Z

$$X^{\zeta,\delta}_{\mathsf{MP}} \;=\; Y(1-W) + WZ,$$

where 1) W is a Bernoulli random variable, such that $W = 0$ iff the algorithm correctly detected the BSCC and estimated transition probabilities within bounds, 2) Y is the value computed by the algorithm if $W = 0$, and the real mean payoff MP when $W = 1$, and 3) Z is any random variable with the range $[0,1]$. The interpretation is as follows: when $W = 0$ we observe the result Y, which has bounded error ζ, and when $W = 1$ we observe arbitrary Z. We note that Y, W, Z are not necessarily independent. By Theorem 4.7 $\mathbb{E}[W] \leq \delta$ and by linearity of expectation: $\mathbb{E}[X^{\zeta,\delta}_{\mathsf{MP}}] = \mathbb{E}[Y] - \mathbb{E}[YW] + \mathbb{E}[WZ]$. For the upper bound, observe that $\mathbb{E}[Y] \leq \mathsf{MP} + \zeta$, $\mathbb{E}[YW]$ is non-negative and $\mathbb{E}[WZ] \leq \delta$. As for the lower bound, note that $\mathbb{E}[Y] \geq \mathsf{MP} - \zeta$, $\mathbb{E}[YW] \leq \delta$ and $\mathbb{E}[WZ]$ is non-negative.

□

As a consequence of Theorem 4.8, if we establish that with $(1-\alpha)$ confidence $X_{\mathsf{MP}}^{\zeta,\delta}$ belongs to the interval $[a,b]$, then we can conclude with $(1-\alpha)$ confidence that MP belongs to the interval $[a-\zeta-\delta, b+\zeta+\delta]$. Standard statistical methods can be applied to find the confidence bound for $X_{\mathsf{MP}}^{\zeta,\delta}$ [Bickel and Doksum 2000].

5. EXPERIMENTAL EVALUATION

We implemented our algorithms in the probabilistic model checker PRISM [Kwiatkowska et al. 2011], and evaluated them on the DTMC examples from the PRISM benchmark suite [Kwiatkowska et al. 2012]. The benchmarks model communication and security protocols, distributed algorithms, and fault-tolerant systems. To demonstrate how our method performs depending on the topology of Markov chains, we also performed experiments on the generic DTMCs shown in Figure 3 and Figure 4, as well as on two CTMCs from the literature that have large BSCCs: "tandem" [Hermanns et al. 1999] and "gridworld" [Younes et al. 2006].

All benchmarks are parametrised by one or more values, which influence their size and complexity, e.g. the number of modelled components. We have made minor modifications to the benchmarks that could not be handled directly by the SMC component of PRISM, by adding self-loops to deadlock states and fixing one initial state instead of multiple.

Our tool can be downloaded at [Daca 2016]. Experiments were done on a Linux 64-bit machine running an AMD Opteron 6134 CPU with a time limit of 15 minutes and a memory limit of 5GB. To increase performance of our tool, we check whether a candidate has been found every 1000 steps; this optimization does not violate correctness of our analysis. See the appendix for a discussion on this bound.

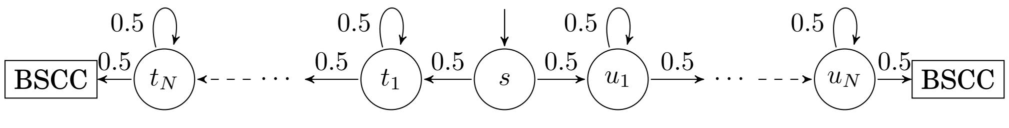


Fig. 4. A Markov chain with two transient parts consisting of N strongly connected singletons, leading to BSCCs with the ring topology of M states.

Reachability. The experimental results for unbounded reachability are shown in Table II. The PRISM benchmarks were checked against their standard properties, when available. We directly compare our method to another topology-agnostic method of [Younes et al. 2010] (SimTermination), where at every step the sampled path is terminated with probability p_{term}. The approach of [Brázdil et al. 2014] with a priori bounds is not included, since it times out even on the smallest benchmarks. In addition, we performed experiments on two methods that are topology-aware: sampling with reachability analysis of [Younes et al. 2010] (SimAnalysis) and the numerical model-checking algorithm of PRISM (MC). The appendix contains detailed experimental evaluation of these methods.

The table shows the size of every example, its minimum probability, the number of BSCCs, and the size of the largest BSCC. Column "time" reports the total wall time for the respective algorithm, and "analysis" shows the time for symbolic reachability analysis in the SimAnalysis method. Highlights show the best result among the topology-agnostic methods. All statistical methods were used with the SPRT test for choosing between the hypothesis, and their results were averaged over five runs.

Table II. Experimental results for unbounded reachability.

Example			BSCC	SimAdaptive	SimTermination[Younes et al. 2010]	SimAnalysis[Younes et al. 2010]		MC
name	size	p_{min}	no., max. size	time	time	time	analysis	time
bluetooth(4)	149K	$7.8 \cdot 10^{-3}$	3K, 1	2.6s	16.4s	83.2s	80.4s	78.2s
bluetooth(7)	569K	$7.8 \cdot 10^{-3}$	5.8K, 1	3.8s	50.2s	284.4s	281.1s	261.2s
bluetooth(10)	>569K	$7.8 \cdot 10^{-3}$	>5.8K, 1	5.0s	109.2s	TO	-	TO
brp(500,500)	4.5M	0.01	1.5K, 1	7.6s	13.8s	35.6s	30.7s	103.0s
brp(2K,2K)	40M	0.01	4.5K, 1	20.4s	17.2s	824.4s	789.9s	TO
brp(10K,10K)	>40M	0.01	>4.5K, 1	89.2s	15.8s	TO	-	TO
crowds(6,15)	7.3M	0.066	>3K, 1	3.6s	253.2s	2.0s	0.7s	19.4s
crowds(7,20)	17M	0.05	>3K, 1	4.0s	283.8s	2.6s	1.1s	347.8s
crowds(8,20)	68M	0.05	>3K, 1	5.6s	340.0s	4.0s	1.9s	TO
eql(15,10)	616G	0.5	1, 1	16.2s	TO	151.8s	145.1s	110.4s
eql(20,15)	1279T	0.5	1, 1	28.8s	TO	762.6s	745.4s	606.6s
eql(20,20)	1719T	0.5	1, 1	31.4s	TO	TO	-	TO
herman(17)	129M	$7.6 \cdot 10^{-6}$	1, 34	23.0s	33.6s	21.6s	0.1s	1.2s
herman(19)	1162M	$1.9 \cdot 10^{-6}$	1, 38	96.8s	134.0s	86.2s	0.1s	1.2s
herman(21)	10G	$4.7 \cdot 10^{-7}$	1, 42	570.0s	TO	505.2s	0.1s	1.4s
leader(6,6)	280K	$2.1 \cdot 10^{-5}$	1, 1	5.0s	5.4s	536.6s	530.3s	491.4s
leader(6,8)	>280K	$3.8 \cdot 10^{-6}$	1, 1	23.0s	26.0s	MO	-	MO
leader(6,11)	>280K	$5.6 \cdot 10^{-7}$	1, 1	153.0s	174.8s	MO	-	MO
nand(50,3)	11M	0.02	51, 1	7.0s	231.2s	36.2s	31.0s	272.0s
nand(60,4)	29M	0.02	61, 1	6.0s	275.2s	60.2s	56.3s	TO
nand(70,5)	67M	0.02	71, 1	6.8s	370.2s	148.2s	144.2s	TO
tandem(500)	>1.7M	$2.4 \cdot 10^{-5}$	1, >501K	2.4s	6.4s	4.6s	3.0s	3.4s
tandem(1K)	1.7M	$9.9 \cdot 10^{-5}$	1, 501K	2.6s	19.2s	17.0s	12.7s	13.0s
tandem(2K)	>1.7M	$4.9 \cdot 10^{-5}$	1, >501K	3.4s	72.4s	62.4s	59.8s	59.4s
gridworld(300)	162M	$1 \cdot 10^{-3}$	598, 89K	8.2s	81.6s	MO	-	MO
gridworld(400)	384M	$1 \cdot 10^{-3}$	798, 160K	8.4s	100.6s	MO	-	MO
gridworld(500)	750M	$1 \cdot 10^{-3}$	998, 250K	5.8s	109.4s	MO	-	MO
Fig.3(16)	37	0.5	1, 1	58.6s	TO	23.4s	0.4s	2.0s
Fig.3(18)	39	0.5	1, 1	TO	TO	74.8.0s	1.8s	2.0s
Fig.3(20)	41	0.5	1, 1	TO	TO	513.6s	11.3s	2.0s
Fig.4(1K,5)	4022	0.5	2, 5	7.8s	218.2s	3.2s	0.5s	1.2s
Fig.4(1K,50)	4202	0.5	2, 50	12.4s	211.8s	3.6s	0.7s	1.0s
Fig.4(1K,500)	6002	0.5	2, 500,	431.0s	218.6s	3.6s	1.0s	1.2s
Fig.4(10K,5)	40K	0.5	2, 5	52.2s	TO	42.2s	25.4s	25.6s
Fig.4(100K,5)	400K	0.5	2, 5	604.2s	5.4s	TO	-	TO

Note. Simulation parameters: $\alpha = \beta = \varepsilon = 0.01$, $\delta = 0.001$, $p_{\mathsf{term}} = 0.0001$. TO means time-out, and MO means memory-out. Our approach is denoted by SimAdaptive here. Highlights show the best result the among topology-agnostic methods.

Finding the optimal termination probability p_{term} for the SimTermination method is a non-trivial task. If the probability is too high, the method might never reach the target states, thus give an incorrect result, and if the value is too low, then it might sample unnecessarily long traces that never reach the target. For instance, to ensure a correct answer on the Markov chain in Figure 3, p_{term} has to decrease exponentially with the number of states. By experimenting we found that the probability $p_{\mathsf{term}} = 0.0001$ is low enough to ensure correct results. See the appendix for experiments with other values of p_{term}.

On most examples our method scales better than the SimTermination method. Our method performs well even on examples with large BSCCs, such as “tandem” and “gridworld,” due to early termination when a goal state is reached. For instance, on the “gridworld” example, most BSCCs do not contain a goal state, thus have to be fully explored, however the probability of reaching such BSCC is low, and as a consequence full BSCC exploration rarely occurs. The SimTermination method performs well when the target states are unreachable or can be reached by short paths. When long paths are necessary to reach the target, the probability that an individual path reaches the target is small, hence many samples are necessary to estimate the real probability with high confidence.

Moreover, it turns out that our method compares well even with methods that have access to the topology of the system. In many cases, the running time of the numerical algorithm MC increases dramatically with the size of the system, while remaining almost constant in our method. The bottleneck of the SimAnalysis algorithm is the reachability analysis of states that cannot reach the target, which in practice can be as difficult as numerical model checking.

LTL and mean payoff. In the second experiment, we compared our algorithm for checking LTL properties and estimating the mean payoff with the numerical methods of PRISM; the results are shown in Table III and IV. We compare against PRISM, since we are not aware of any SMC-based or topology-agnostic approach for mean payoff, or full LTL. For mean payoff, we computed 95%-confidence bound of size 0.22 with parameters $\delta = 0.011, \zeta = 0.08$, and for LTL we used the same parameters as for reachability. We report results only on a single model of each type, where either method did not time out. In general our method scales better when BSCCs are fairly small and are discovered quickly.

Table III. Experimental results for LTL properties.

Example		LTL	
name	property	SimAdaptive time	MC time
bluetooth(10)	$\Box\Diamond$	8.0s	TO
brp(10K,10K)	$\Diamond\Box$	90.0s	TO
crowds(8,20)	$\Diamond\Box$	9.0s	TO
eql(20,20)	$\Box\Diamond$	7.0s	MO
herman(21)	$\Box\Diamond$	TO	2.0s
leader(6,5)	$\Box\Diamond$	277.0s	117.0s
nand(70,5)	$\Box\Diamond$	4.0s	TO
tandem(2K)	$\Box\Diamond$	TO	221.0s
gridworld(100)	$\Box\Diamond \to \Diamond\Box$	TO	110.4s
Fig.3(20)	$\Box\Diamond \to \Box\Diamond$	TO	
Fig.4(100K,5)	$\Box\Diamond$	348.0s	TO
Fig.4(1K,500)	$\Box\Diamond$	827.0s	2.0s

Note. Simulation parameters for LTL: $\alpha = \beta = \varepsilon = 0.01$, $\delta = 0.001$.

Table IV. Experimental results for mean-payoff properties.

Example name	Mean payoff SimAdaptive time	Mean payoff MC time
bluetooth(10)	3.0s	TO
brp(10K,10K)	6.6s	TO
crowds(8,20)	2.0s	TO
eql(20,20)	2.6s	TO
herman(21)	MO	3.0s
leader(6,6)	48.5	576.0
nand(70,5)	2.0s	294.0s
tandem(500)	TO	191.0s
gridworld(50)	TO	58.1s
Fig.3(20)	TO	1.8s
Fig.4(100K,5)	79.6s	TO
Fig.4(1K,500)	TO	2.0s

Note. For mean-payoff we computed a 95%-confidence interval of size 0.22 with $\delta = 0.011, \zeta = 0.08$.

6. DISCUSSION

As demonstrated by the experimental results, our method is fast on systems that are (1) shallow, and (2) with small BSCCs. In such systems, the BSCC is reached quickly and the candidate is built-up quickly. Further, recall that the BSCC is reported when a k-candidate is found, and that k is increased with each candidate along the path. Hence, when there are many strongly connected sets, and thus many candidates, the BSCC is detected by a k-candidate for a large k. However, since k grows linearly in the number of candidates, the most important and limiting factor is the size of BSCCs.

We state the dependency on the depth of the system and BSCC sizes formally. We pick $\delta := \frac{\varepsilon}{2}$ and let

$$sim = \frac{-\log \frac{\beta}{1-\alpha} \log \frac{1-\beta}{\alpha}}{\log \frac{p-\varepsilon+\delta}{p+\varepsilon-\delta} \log \frac{1-p-\varepsilon+\delta}{1-p+\varepsilon-\delta}} \qquad \text{and} \qquad k_i = \frac{i - \log \delta}{-\log(1-p_{\mathsf{min}})}$$

denote the a priori upper bound on the number of simulations necessary for the SPRT (cf. Section 2.2) and the strength of candidates as in Algorithm 2, respectively.

THEOREM 6.1. *Let R denote the expected number of steps before reaching a BSCC and B the maximum size of a BSCC. Further, let*

$$T := \max_{C \in \mathsf{BSCC}; s,s' \in C} \mathbb{E}[\textit{time to reach } s' \textit{ from } s] .$$

In particular, $T \in \mathcal{O}(B/p_{\mathsf{min}}^B)$. Then the expected running time of Algorithms 2 and 3 is at most

$$\mathcal{O}(sim \cdot k_{R+B} \cdot B \cdot T) .$$

PROOF. We show that the expected running time of each simulation is at most $k_{R+B} \cdot B \cdot T$. Since the expected number of states visited is bounded by $R + B$, the expected number of candidates on a run is less than $2(R + B) - 1$. Since k_i grows linearly in i it is sufficient to prove that the expected time to visit each state of a BSCC once (when starting in BSCC) is at most $B \cdot T$. We order the states of a BSCC as $s_1, \ldots, s_b$, then the time is at most $\sum_{i=1}^{b} T$, where $b \leq B$. This yields the result since $R \in \mathcal{O}(k_{R+B} \cdot B \cdot T)$.

It remains to prove that $T \leq B/p_{\mathsf{min}}^B$. Let s be a state of a BSCC of size at most B. Then, for any state s' from the same BSCC, the shortest path from s to s' has length at most B and probability at least p_{min}^B. Consequently, if starting at s, we haven't reached s' after B steps with probability at most $1 - p_{\mathsf{min}}^B$, and we are instead in some state

$s'' \neq s'$, from which, again, the probability to reach s' within B steps at least p_{min}^B. Hence, the expected time to reach s' from s is at most

$$\sum_{i=1}^{\infty} B \cdot i(1 - p_{\mathsf{min}}^B)^{i-1} p_{\mathsf{min}}^B,$$

where i indicates the number of times a sequence of B steps is observed. The series can be summed by differentiating a geometric series. As a result, we obtain a bound B/p^B.

□

Systems that have large deep BSCCs require longer time to reach for the required level of confidence. However, such systems are often difficult to handle also for other methods agnostic of the topology. For instance, correctness of [Younes et al. 2010] on the example in Figure 3 relies on the termination probability p_{term} being at most $1 - \lambda$, which is less than 2^{-n} here. Larger values lead to incorrect results and smaller values to paths of exponential length. Nevertheless, our procedure usually runs faster than the bound suggest; for detailed discussion see the appendix.

7. CONCLUSION

To the best of our knowledge, we propose the first statistical model-checking method that exploits the information provided by each simulation run on the fly, in order to detect quickly a potential BSCC, and verify LTL properties with the desired confidence. This is also the first application of SMC to quantitative properties such as mean payoff. We note that for our method to work correctly, the precise value of p_{min} is not necessary, but a lower bound is sufficient. This lower bound can come from domain knowledge, or can be inferred directly from description of white-box systems, such as the PRISM benchmark.

The approach we present is not meant to replace the other methods, but rather to be an addition to the repertoire of available approaches. Our method is particularly valuable for models that have small BSCCs and huge state space, such as many of the PRISM benchmarks.

In future work, we plan to investigate the applicability of our method to Markov decision processes, and to deciding language equivalence between two Markov chains. The idea of guessing BSCCs by simulation has already been re-used in order to estimate distances between Markov chains [Daca et al. 2016b]

APPENDIX

A. DETAILED EXPERIMENTS

Table V shows detailed experimental result for unbounded reachability. Compared to Table II we included: 1) experiments for the SimTermination method with two other values of p_{term}, 2) we report the number of sampled paths as "samples," and 3) we report the average length of sampled paths as "path length." Topology-agnostic methods, such as SimAdaptive and SimTermination, cannot be compared directly with topology-aware methods, such as SimAnalysis and MC, however for reader's curiosity we highlighted in the table the best results among *all* methods.

We observed that in the "herman" example the SMC algorithms work unusually slow. This problem seems to be caused by a bug in the original sampling engine of PRISM and it appears that all SMC algorithms suffer equally from this problem.

Table V. Detailed experimental results for unbounded reachability.

Example	SimAdaptive			SimTermination, $p_{\text{term}} = 10^{-3}$			SimTermination, $p_{\text{term}} = 10^{-4}$			SimTermination, $p_{\text{term}} = 10^{-5}$			SimAnalysis				MC
name	time	samples	path length	time	samples	path length	time	samples	path length	time	samples	path length	time	samples	path length	analysis	time
bluetooth(4)	2.6s	243	499	185.0s	43K	387	16.4s	3389	484	6.4s	463	495	83.2s	219	502	80.4s	78.2s
bluetooth(7)	3.8s	243	946	697.4s	106K	604	50.2s	6480	897	10.2s	792	931	284.4s	219	937	281.1s	261.2s
bluetooth(10)	5.0s	243	1391	TO	-	-	109.2s	9827	1292	15.0s	932	1380	TO	-	-	-	TO
brp(500,500)	7.6s	230	3999	3.2s	258	963	13.8s	258	9758	107.2s	258	104K	35.6s	207	3045	30.7s	103.0s
brp(2K,2K)	20.4s	230	13K	3.4s	258	1029	17.2s	258	9127	115.0s	258	98K	824.4s	207	12K	789.9s	TO
brp(10K,10K)	89.2s	230	62K	3.6s	258	960	15.8s	258	10K	109.4s	258	96K	TO	-	-	-	TO
crowds(6,15)	3.6s	395	879	29.2s	7947	878	253.2s	7477	8735	TO	-	-	2.0s	400	85	0.7s	19.4s
crowds(7,20)	4.0s	485	859	32.6s	9378	850	283.8s	8993	8464	TO	-	-	2.6s	473	98	1.1s	347.8s
crowds(8,20)	5.6s	830	824	38.2s	11K	821	340.0s	10K	8132	TO	-	-	4.0s	793	110	1.9s	TO
eql(15,10)	16.2s	1149	652	303.2s	28K	628	TO	-	-	TO	-	-	151.8s	1100	201	145.1s	110.4s
eql(20,15)	28.8s	1090	1299	612.8s	44K	723	TO	-	-	TO	-	-	762.6s	999	347	745.4s	606.6s
eql(20,20)	31.4s	1071	1401	TO	11K	156	TO	-	-	TO	-	-	TO	-	-	-	TO
herman(17)	23.0s	243	30	257.6s	2101	30	33.6s	381	32	29.0s	279	31	21.6s	219	30	0.1s	1.2s
herman(19)	96.8s	243	40	TO	-	-	134.0s	355	38	254.4s	279	40	86.2s	219	38	0.1s	1.2s
herman(21)	570.0s	243	46	MO	-	-	TO	-	-	MO	-	-	505.2s	219	48	0.1s	1.4s
leader(6,6)	5.0s	243	7	7.6s	437	7	5.4s	258	7	5.0s	258	7	536.6s	219	7	530.3s	491.4s
leader(6,8)	23.0s	243	7	62.4s	560	7	26.0s	279	7	26.2s	258	7	MO	-	-	-	MO
leader(6,11)	153.0s	243	7	TO	-	-	174.8s	279	7	776.8s	258	7	MO	-	-	-	MO
nand(50,3)	7.0s	899	1627	570.6s	140K	846	231.2s	21K	4632	TO	-	-	36.2s	1002	1400	31.0s	272.0s
nand(60,4)	6.0s	522	2431	TO	-	-	275.2s	25K	4494	TO	-	-	60.2s	458	2160	56.3s	TO
nand(70,5)	6.8s	391	3343	TO	-	-	370.2s	30K	4643	TO	-	-	148.2s	308	3080	144.2s	TO
tandem(500)	2.4s	243	501	59.6s	43K	394	6.4s	3318	489	2.0s	412	500	4.6s	219	501	3.0s	3.4s
tandem(1K)	2.6s	243	1001	328.4s	114K	632	19.2s	6932	954	3.4s	858	995	17.0s	219	1001	12.7s	13.0s
tandem(2K)	3.4s	243	2001	TO	-	-	72.4s	14K	1811	6.6s	1093	1985	62.4s	219	2001	59.8s	59.4s
gridworld(300)	8.2s	1187	453	214.4s	46K	349	81.6s	18K	437	77.4s	16K	449	MO	-	-	-	MO
gridworld(400)	8.4s	1047	543	274.8s	53K	399	100.6s	18K	531	93.0s	16K	548	MO	-	-	-	MO
gridworld(500)	5.8s	480	637	277.4s	57K	431	109.4s	18K	605	104.4s	15K	627	MO	-	-	-	MO
Fig.3(16)	58.6s	128	140K	TO	-	-	TO	-	-	TO	-	-	23.4s	115	141K	0.4s	2.0s
Fig.3(18)	TO	-	-	2.8s	258	1015	TO	-	-	TO	-	-	74.8s	115	537K	1.8s	2.0s
Fig.3(20)	TO	-	-	WRONG	-	-	TO	-	-	TO	-	-	513.6s	119	2M	11.3s	2.0s
Fig.4(1K,5)	7.8s	1109	2489	TO	-	-	218.2s	23K	5916	TO	-	-	3.2s	896	1027	0.5s	1.2s
Fig.4(1K,50)	12.4s	1115	4306	TO	-	-	211.8s	23K	5880	TO	-	-	3.6s	881	1037	0.7s	1.0s
Fig.4(1K,500)	431.0s	1002	177K	TO	-	-	218.6s	23K	5903	TO	-	-	3.6s	968	1042	1.0s	1.2s
Fig.4(10K,5)	52.2s	1161	20K	2.6s	258	1072	TO	-	-	TO	-	-	42.2s	1057	10K	25.4s	25.6s
Fig.4(100K,5)	604.2s	1331	200K	2.6s	258	981	5.4s	258	9939	TO	-	-	TO	-	-	-	TO

Note. Simulation parameters: $\alpha = \beta = \varepsilon = 0.01$, $\delta = 0.001$. TO means a timeout or memory out, and WRONG means that the reported result was incorrect, due to p_{term} being too large. Our approach is denoted by SimAdaptive here. Highlights show the best result among *all* methods.

B. IMPLEMENTATION DETAILS

In our algorithms we frequently check whether the simulated path contains a candidate with the required strength. To reduce the time needed for this operation we use two optimisation: 1) we record SCs visited on the path, 2) we check if a candidate has been found every $C_b \geq 1$ steps. Our data structure records the sequence of SCs that have been encountered on the simulated path. The candidate of the path is then the last SC in the sequence. We also record the number of times each state in the candidate has been encountered. By using this data structure we avoid traversing the entire path every time we check if a strong k-candidate has been reached.

To further reduce the overhead, we update our data structure every C_b steps (in our experiments $C_b = 1000$). Figures 5 and 6 show the impact of C_b on the running time for two Markov chains. The optimal value of C_b varies among examples, however experience shows that $C_b \approx 1000$ is a reasonable choice.

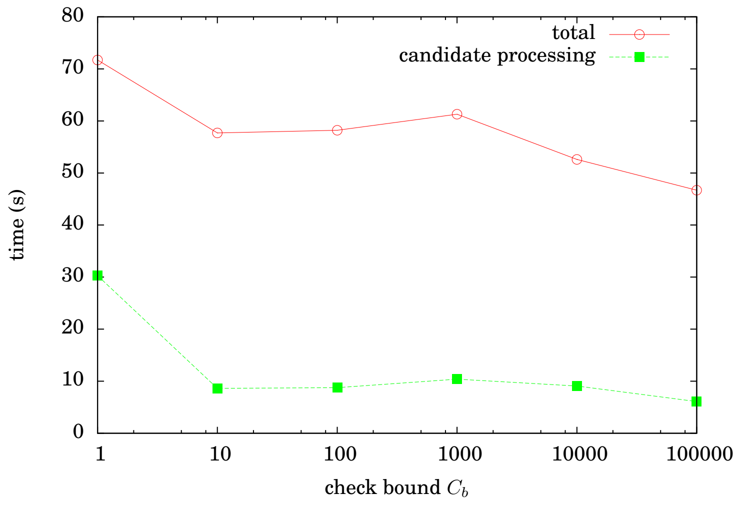


Fig. 5. Total running time and time for processing candidates for a Markov chain in Figure 3 depending on the check bound C_b.

C. THEORETICAL VS. EMPIRICAL RUNNING TIME

In this section, we compare the theoretical upper bound on running time given in Theorem 6.1 to empirical data. We omit the number of simulation runs (term sim in the theorem), and report only the logarithm of average simulation length. Figures 7, 8 and 9 present the comparison for different topologies of Markov chains. In Figure 7 we present the comparison for the worst-case Markov chain, which requires the longest paths to discover the BSCCs as a k-candidate. This Markov chain is like the one in Figure 3, but where the last state has a single outgoing transition to the initial state. Figure 8 suggests that the theoretical bound can be a good predictor of running time with respect to the depth of the system, however, Figure 9 shows that it is very conservative with respect to the size of BSCCs.

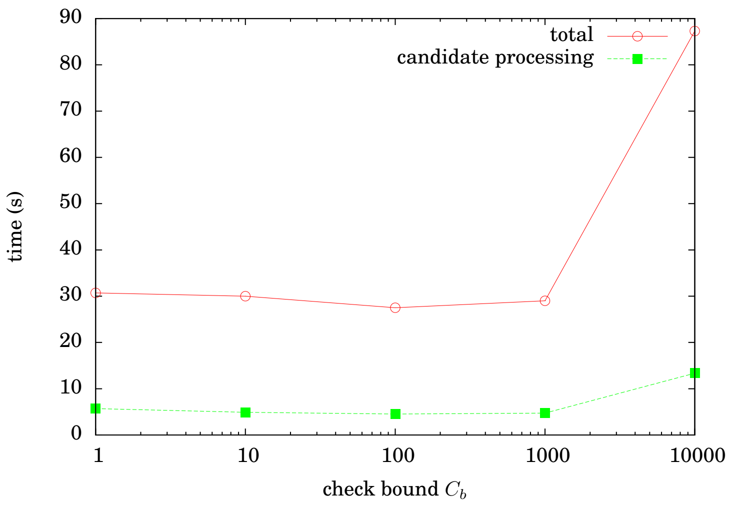


Fig. 6. Total running time and time for processing candidates for the "eql(20,20)" benchmark depending on the check bound C_b.

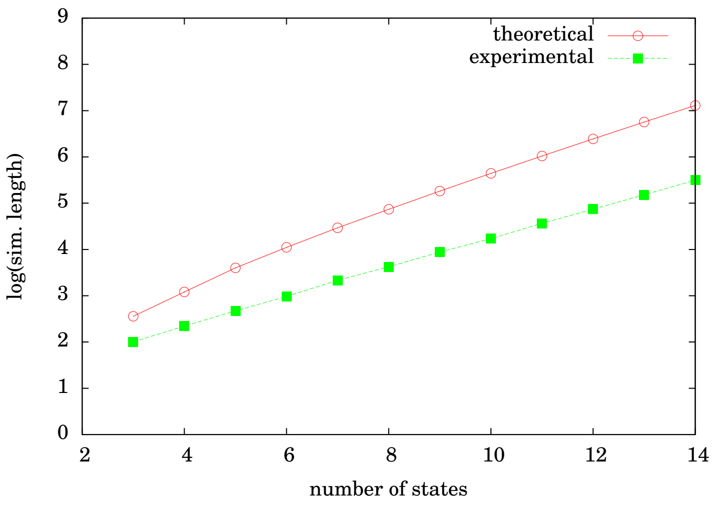


Fig. 7. Average length of simulations for a Markov chain like in Figure 3, but where the last state has a single outgoing transition to the initial state.

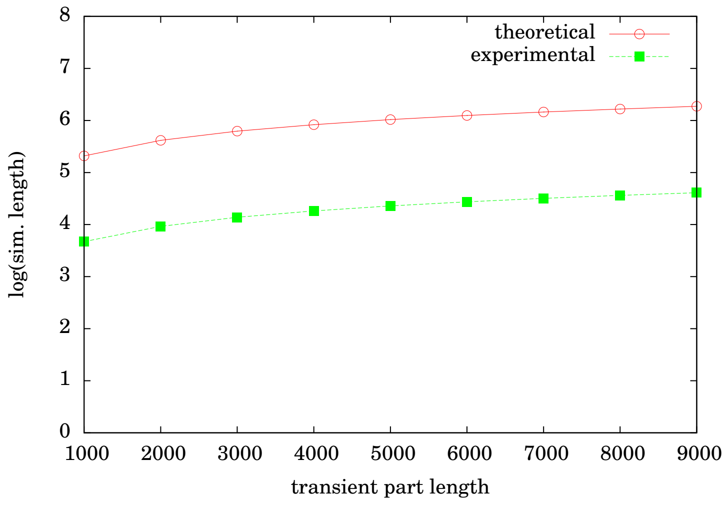


Fig. 8. Average length of simulations for the MC in Figure 4, where $M = 5$ and N varies.

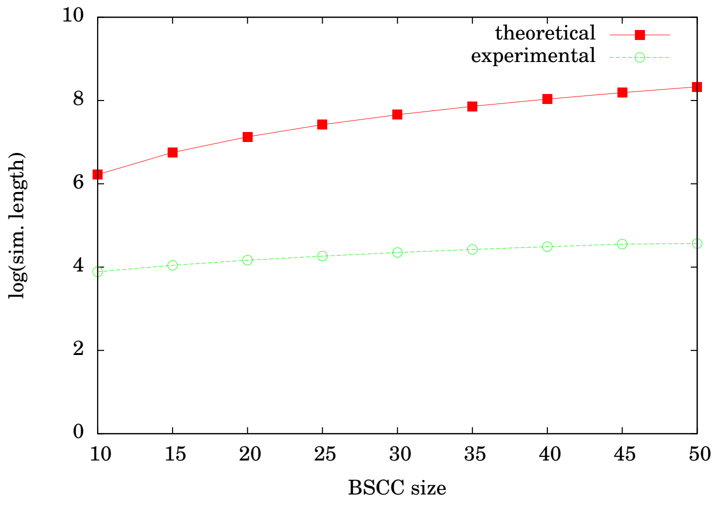


Fig. 9. Average length of simulations for the MC in Figure 4, where $N = 1000$ and M varies.

REFERENCES

Christel Baier and Joost-Pieter Katoen. 2008. *Principles of model checking*. MIT Press.

P.J. Bickel and K.A. Doksum. 2000. *Mathematical statistics: basic ideas and selected topics*. Number Bd. 1 in Mathematical Statistics: Basic Ideas and Selected Topics. Prentice Hall. http://books.google.at/books?id=8poZAQAAIAAJ

Tomáš Brázdil, Krishnendu Chatterjee, Martin Chmelík, Vojtěch Forejt, Jan Křetínský, Marta Z. Kwiatkowska, David Parker, and Mateusz Ujma. 2014. Verification of Markov Decision Processes Using Learning Algorithms. In *ATVA*. 98–114.

Peter E. Bulychev, Alexandre David, Kim Guldstrand Larsen, Marius Mikucionis, Danny Bøgsted Poulsen, Axel Legay, and Zheng Wang. 2012. UPPAAL-SMC: Statistical Model Checking for Priced Timed Automata. In *QAPL*. 1–16.

Krishnendu Chatterjee. 2012. Robustness of structurally equivalent concurrent parity games. In *FoSSaCS*. Springer, 270–285.

Przemysław Daca. 2016. Tool for the paper. http://pub.ist.ac.at/~przemek/pa_tool.html. (2016).

Przemysław Daca, Thomas A. Henzinger, Jan Kretínský, and Tatjana Petrov. 2016a. Faster Statistical Model Checking for Unbounded Temporal Properties. In *TACAS*. 112–129.

Przemyslaw Daca, Thomas A. Henzinger, Jan Kretínský, and Tatjana Petrov. 2016b. Linear Distances between Markov Chains. In *27th International Conference on Concurrency Theory, CONCUR 2016, August 23-26, 2016, Québec City, Canada (LIPIcs)*, Josée Desharnais and Radha Jagadeesan (Eds.), Vol. 59. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 20:1–20:15. DOI:http://dx.doi.org/10.4230/LIPIcs.CONCUR.2016.20

Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikucionis, and Danny Bøgsted Poulsen. 2015. Uppaal SMC tutorial. *STTT* 17, 4 (2015), 397–415.

Radu Grosu and Scott A. Smolka. 2005. Monte Carlo Model Checking. In *TACAS*. 271–286.

Ru He, Paul Jennings, Samik Basu, Arka P. Ghosh, and Huaiqing Wu. 2010. A bounded statistical approach for model checking of unbounded until properties. In *ASE*. 225–234.

Thomas Hérault, Richard Lassaigne, Frédéric Magniette, and Sylvain Peyronnet. 2004. Approximate Probabilistic Model Checking. In *VMCAI*. 73–84.

Holger Hermanns, Joachim Meyer-Kayser, and Markus Siegle. 1999. Multi terminal binary decision diagrams to represent and analyse continuous time Markov chains. In *3rd Int. Workshop on the Numerical Solution of Markov Chains*. Citeseer, 188–207.

Cyrille Jégourel, Axel Legay, and Sean Sedwards. 2012. A Platform for High Performance Statistical Model Checking - PLASMA. In *TACAS*. 498–503.

Sumit Kumar Jha, Edmund M. Clarke, Christopher James Langmead, Axel Legay, André Platzer, and Paolo Zuliani. 2009. A Bayesian Approach to Model Checking Biological Systems. In *CMSB*. 218–234.

Marta Z. Kwiatkowska, Gethin Norman, and David Parker. 2011. PRISM 4.0: Verification of Probabilistic Real-Time Systems. In *CAV*. 585–591.

Marta Z. Kwiatkowska, Gethin Norman, and David Parker. 2012. The PRISM Benchmark Suite. In *QEST*. 203–204.

Richard Lassaigne and Sylvain Peyronnet. 2008. Probabilistic verification and approximation. *Ann. Pure Appl. Logic* 152, 1-3 (2008), 122–131.

James R Norris. 1998. *Markov chains*. Cambridge university press.

Johan Oudinet, Alain Denise, Marie-Claude Gaudel, Richard Lassaigne, and Sylvain Peyronnet. 2011. Uniform Monte-Carlo Model Checking. In *FASE*. 127–140.

Amir Pnueli. 1977. The temporal logic of programs. In *FOCS*. 46–57.

Diana El Rabih and Nihal Pekergin. 2009. Statistical Model Checking Using Perfect Simulation. In *ATVA*. 120–134.

Koushik Sen, Mahesh Viswanathan, and Gul Agha. 2004. Statistical Model Checking of Black-Box Probabilistic Systems. In *CAV*. 202–215.

Koushik Sen, Mahesh Viswanathan, and Gul Agha. 2005. On Statistical Model Checking of Stochastic Systems. In *CAV*. 266–280.

Eilon Solan. 2003. Continuity of the value of competitive Markov decision processes. *Journal of Theoretical Probability* 16, 4 (2003), 831–845.

Abraham Wald. 1945. Sequential tests of statistical hypotheses. *The Annals of Mathematical Statistics* 16, 2 (1945), 117–186.

Håkan L. S. Younes. 2004. Planning and verification for stochastic processes with asynchronous events. In *AAAI*. 1001–1002.

Håkan L. S. Younes, Edmund M. Clarke, and Paolo Zuliani. 2010. Statistical Verification of Probabilistic Properties with Unbounded Until. In *SBMF*. 144–160.

Håkan L. S. Younes, Marta Z. Kwiatkowska, Gethin Norman, and David Parker. 2006. Numerical vs. statistical probabilistic model checking. *STTT* 8, 3 (2006), 216–228.

Håkan L. S. Younes and Reid G. Simmons. 2002. Probabilistic Verification of Discrete Event Systems using Acceptance Sampling. In *CAV*. Springer, 223–235.

Håkan L. S. Younes and Reid G. Simmons. 2006. Statistical probabilistic model checking with a focus on time-bounded properties. *Inf. Comput.* 204, 9 (2006), 1368–1409.

From LTL and Limit-Deterministic Büchi Automata to Deterministic Parity Automata *

Javier Esparza[1], Jan Křetínský[1], Jean-François Raskin[2], and Salomon Sickert[1]

[1] Technische Universität München {esparza, jan.kretinsky, sickert}@in.tum.de
[2] Université libre de Bruxelles jraskin@ulb.ac.be

Abstract. Controller synthesis for general linear temporal logic (LTL) objectives is a challenging task. The standard approach involves translating the LTL objective into a deterministic parity automaton (DPA) by means of the Safra-Piterman construction. One of the challenges is the size of the DPA, which often grows very fast in practice, and can reach double exponential size in the length of the LTL formula. In this paper we describe a single exponential translation from limit-deterministic Büchi automata (LDBA) to DPA, and show that it can be concatenated with a recent efficient translation from LTL to LDBA to yield a double exponential, "Safraless" LTL-to-DPA construction. We also report on an implementation, a comparison with the SPOT library, and performance on several sets of formulas, including instances from the 2016 SyntComp competition.

1 Introduction

Limit-deterministic Büchi automata (LDBA, also known as semi-deterministic Büchi automata) were introduced by Courcoubetis and Yannakakis (based on previous work by Vardi) to solve the qualitative probabilistic model-checking problem: Decide if the executions of a Markov chain or Markov Decision Process satisfy a given LTL formula with probability 1 [Var85,VW86,CY95]. The problem faced by these authors was that fully nondeterministic Büchi automata (NBAs), which are as expressible as LTL, and more, cannot be used for probabilistic model checking, and deterministic Büchi automata (DBA) are less expressive than LTL. The solution was to introduce LDBAs as a model in-between: as expressive as NBAs, but deterministic enough.

After these papers, LDBAs received little attention. The alternative path of translating the LTL formula into an equivalent fully deterministic Rabin automaton using Safra's construction [Saf88] was considered a better option, mostly because it also solves the quantitative probabilistic model-checking problem (computing the probability of the executions that satisfy a formula). However, recent papers have shown that LDBAs were unjustly forgotten. Blahoudek *et al.* have

* This work is partially funded by the DFG Research Training Group "PUMA: Programm- und Modell-Analyse" (GRK 1480), DFG project "Verified Model Checkers", the ERC Starting Grant (279499: inVEST), and the Czech Science Foundation, grant No. P202/12/G061.

shown that LDBAs are easy to complement [BHS$^+$16]. Kini and Viswanathan have given a single exponential translation of $\mathrm{LTL}_{\setminus \mathbf{GU}}$ to LDBA [KV15]. Finally, Sickert *et al.* describe in [SEJK16] a double exponential translation for full LTL that can also be applied to the quantitative case, and behaves better than Safra's construction in practice.

In this paper we add to this trend by showing that LDBAs are also attractive for synthesis. The standard solution to the synthesis problem with LTL objectives consists of translating the LTL formula into a deterministic parity automaton (DPA) with the help of the Safra-Piterman construction [Pit07]. While limit-determinism is not "deterministic enough" for the synthesis problem, we introduce a conceptually simple and worst-case optimal translation LDBA$\rightarrow$DPA. Our translation bears some similarities with that of [Fin15] where, however, a Muller acceptance condition is used. This condition can also be phrased as a Rabin condition, but not as a parity condition. Moreover, the way of tracking all possible states and finite runs differs.

Together with the translation LTL$\rightarrow$LDBA of [SEJK16], our construction provides a "Safraless", procedure to obtain a DPA from an LTL formula. However, the direct concatenation of the two constructions does not yield an algorithm of optimal complexity: the LTL$\rightarrow$LDBA translation is double exponential (and there is a double-exponential lower bound), and so for the LTL$\rightarrow$DPA translation we only obtain a triple exponential bound. In the second part of the paper we solve this problem. We show that the LDBAs derived from LTL formulas satisfy a special property, and prove that for such automata the concatenation of the two constructions remains double exponential. To the best of our knowledge, this is the first double exponential "Safraless" LTL$\rightarrow$DPA procedure. (Another asymptotically optimal "Safraless" procedure for determinization of Büchi automata with Rabin automata as target has been presented in [FKVW15].)

In the third and final part, we report on the performance of an implementation of our LTL$\rightarrow$LDBA$\rightarrow$DPA construction, and compare it with algorithms implemented in the SPOT library [DLLF$^+$16]. Note that it is not possible to force SPOT to always produce DPA, sometimes it produces a deterministic generalized Büchi automaton (DGBA). The reason is that DGBA are often smaller than DPA (if they exist) and game-solving algorithms for DGBA are not less efficient than for DPA. Therefore, also our implementation may produce DGBA in some cases. We show that our implementation outperforms SPOT for several sets of parametric formulas and formulas used in synthesis examples taken from the SyntComp 2016 competition, and remains competitive for randomly generated formulas.

Structure of the paper Section 2 introduces the necessary preliminaries about automata. Section 3 defines the translation LDBA$\rightarrow$DPA. Section 4 shows how to compose of LTL$\rightarrow$LDBA and LDBA$\rightarrow$DPA in such a way that the resulting DPA is at most doubly exponential in the size of the LTL formula. Section 5 reports on the experimental evaluation of this worst-case optimal translation, and Section 6 contains our conclusions. Several proofs and more details on the implementation can be found in [EKRS17].

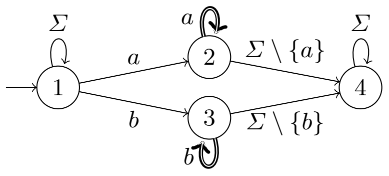


Fig. 1: An LDBA for the LTL language $\mathbf{FG}a \vee \mathbf{FG}b$. The behavior of A is deterministic within the subset of states $Q_d = \{2, 3, 4\}$ which is a trap, the set of accepting transitions are depicted in bold face and they are defined only between states of Q_d.

2 Preliminaries

Büchi automata A (nondeterministic) ω-word automaton A with Büchi acceptance condition (NBA) is a tuple $(Q, q_0, \Sigma, \delta, \alpha)$ where Q is a finite set of states, $q_0 \in Q$ is the *initial* state, Σ is a finite alphabet, $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation, and $\alpha \subseteq \delta$ is the set of *accepting* transitions[3]. W.l.o.g. we assume that δ is total in the following sense: for all $q \in Q$, for all $\sigma \in \Sigma$, there exists $q' \in Q$ such that $(q, \sigma, q') \in \delta$. A is *deterministic* if for all $q \in Q$, for all $\sigma \in \Sigma$, there exists a unique $q' \in Q$ such that $(q, \sigma, q') \in \delta$. When δ is deterministic and total, it can be equivalently seen as a function $\delta : Q \times \Sigma \to Q$. Given $S \subseteq Q$ and $\sigma \in \Sigma$, let $\mathsf{post}^{\sigma}_{\delta}(S) = \{q' \mid \exists q \in S \cdot (q, \sigma, q') \in \delta\}$.

A *run* of A on a ω-word $w : \mathbb{N} \to \Sigma$ is a ω-sequence of states $\rho : \mathbb{N} \to Q$ such that $\rho(0) = q_0$ and for all positions $i \in \mathbb{N}$, we have that $(\rho(i), w(i), \rho(i+1)) \in \delta$. A run ρ is *accepting* if there are infinitely many positions $i \in \mathbb{N}$ such that $(\rho(i), w(i), \rho(i+1)) \in \alpha$. The *language* defined by A, denoted by $\mathsf{L}(A)$, is the set of ω-words w for which A has an accepting run.

A *limit-deterministic Büchi automaton* (LDBA) is a Büchi automaton $A = (Q, q_0, \Sigma, \delta, \alpha)$ such that there exists a subset $Q_d \subseteq Q$ satisfying the three following properties:

1. $\alpha \subseteq Q_d \times \Sigma \times Q_d$, i.e. all accepting transitions are transitions within Q_d;
2. $\forall q \in Q_d \cdot \forall \sigma \in \Sigma \cdot \forall q_1, q_2 \in Q \cdot (q, \sigma, q_1) \in \delta \wedge (q, \sigma, q_2) \in \delta \to q_1 = q_2$, i.e. the transition relation δ is deterministic within Q_d
3. $\forall q \in Q_d \cdot \forall \sigma \in \Sigma \cdot \forall q' \in Q \cdot (q, \sigma, q') \in \delta \to q' \in Q_d$, i.e. Q_d is a trap (when Q_d is entered it is never left).

W.l.o.g. we assume that $q_0 \in Q \setminus Q_d$, and we denote $Q \setminus Q_d$ by $\overline{Q_d}$. Courcoubetis and Yannakakis show that for every ω-regular language $\mathcal{L}$, there exists an LDBA A such that $\mathsf{L}(A) = \mathcal{L}$ [CY95]. That is, LDBAs are as expressive as NBAs. An example of LDBA is given in Fig. 1. Note that the language accepted by this LDBA cannot be recognized by a deterministic Büchi automaton.

[3] Here, we consider automata on infinite words with acceptance conditions based on transitions. It is well known that there are linear translations from automata with acceptance conditions defined on transitions to automata with acceptance conditions defined on states, and vice-versa.

Parity automata A deterministic ω-word automaton A with *parity* acceptance condition (DPA) is a tuple $(Q, q_0, \Sigma, \delta, p)$, defined as for deterministic Büchi automata with the exception of the acceptance condition p, which is now a function assigning an integer in $\{1, 2, \ldots, d\}$, called a *color*, to each transition in the automaton. Colors are naturally ordered by the order on integers.

Given a run ρ over a word w, the infinite sequence of colors traversed by the run ρ is noted $p(\rho)$ and is equal to $p(\rho(0), w(0), \rho(1))\ p((\rho(1), w(1), \rho(2)) \ldots p(\rho(n), w(n), \rho(n+1)) \ldots$. A run ρ is *accepting* if the minimal color that appears infinitely often along $p(\rho)$ is *even*. The *language* defined by A, denoted by $\mathsf{L}(A)$ is the set of ω-words w for which A has an accepting run.

While deterministic Büchi automata are not expressively complete for the class of ω-regular languages, DPAs are complete for ω-regular languages: for every ω-regular language $\mathcal{L}$ there exists a DPA A such that $\mathsf{L}(A) = \mathcal{L}$, see e.g. [Pit07].

3 From LDBA to DPA

3.1 Run DAGs and their coloring

Run DAG A nondeterministic automaton A may have several (even an infinite number of) runs on a given ω-word w. As in [KV01], we represent this set of runs by means of a directed acyclic graph structure called the *run DAG* of A on w. Given an LDBA $A = (Q, Q_d, q_0, \Sigma, \delta, \alpha)$, this graph $G_w = (V, E)$ has a set of vertices $V \subseteq Q \times \mathbb{N}$ and edges $E \subseteq V \times V$ defined as follows:

- $V = \bigcup_{i \in \mathbb{N}} V_i$, where the sets V_i are defined inductively:
  - $V_0 = \{(q_0, 0)\}$, and for all $i \geq 1$,
  - $V_i = \{(q, i) \mid \exists (q', i-1) \in V_{i-1} : (q', w(i), q) \in \delta\}$;
- $E = \{((q, i), (q', i+1)) \in V_i \times V_{i+1} \mid (q, w(i), q') \in \delta\}$.

We denote by V_i^d the set $V_i \cap (Q_d \times \{i\})$ that contains the subset of vertices of layer i that are associated with states in Q_d.

Observe that all the paths of G_w that start from $(q_0, 0)$ are runs of A on w, and, conversely, each run ρ of A on w corresponds exactly to one path in G_w that starts from $(q_0, 0)$. So, we call *runs* the paths in the run DAG G_w. In particular, we say that an infinite path $v_0 v_1 \ldots v_n \ldots$ of G_w is an accepting run if there are infinitely many positions $i \in \mathbb{N}$ such that $v_i = (q, i)$, $v_{i+1} = (q', i+1)$, and $(q, w(i), q') \in \alpha$. Clearly, w is accepted by A if and only if there is an accepting run in G_w. We denote by $\rho(0..n) = v_0 v_1 \ldots v_n$ the prefix of length $n+1$ of the run ρ.

Ordering of runs A function $\mathsf{Ord} : Q \to \{1, 2, \ldots, |Q_d|, +\infty\}$ is called an *ordering* of the states of A w.r.t. Q_d if Ord defines a strict total order on the state from Q_d, and maps each state $q \in \overline{Q_d}$ to $+\infty$, i.e.:

- for all $q \in \overline{Q_d}$, $\mathsf{Ord}(q) = +\infty$,
- for all $q \in Q_d$, $\mathsf{Ord}(q) \neq +\infty$, and

– for all $q, q' \in Q_d$, $\mathsf{Ord}(q) = \mathsf{Ord}(q')$ implies $q = q'$.

We extend Ord to vertices in G_w as follows: $\mathsf{Ord}((q,i)) = \mathsf{Ord}(q)$.

Starting from Ord, we define the following pre-order on the set of run prefixes of the run DAG G_w. Let $\rho(0..n) = v_0 v_1 \dots v_n \dots$ and $\rho'(0..n) = v'_0 v'_1 \dots v'_n \dots$ be two run prefixes of length $n+1$, we write $\rho(0..n) \sqsubseteq \rho'(0..n)$, if $\rho(0..n)$ is *smaller than* $\rho'(0..n)$, which is defined as:

– for all i, $0 \leq i \leq n$, $\mathsf{Ord}(\rho(i)) = \mathsf{Ord}(\rho'(i))$, or
– there exists i, $0 \leq i \leq n$, such that:
  - $\mathsf{Ord}(\rho(i)) < \mathsf{Ord}(\rho'(i))$, and
  - for all j, $0 \leq j < i$, $\mathsf{Ord}(\rho(j)) = \mathsf{Ord}(\rho'(j))$.

This is extended to (infinite) runs as: $\rho \sqsubseteq \rho'$ iff for all $i \geq 0 \cdot \mathsf{Ord}(\rho(0..i)) \sqsubseteq \mathsf{Ord}(\rho'(0..i))$.

Remark 1. If A accepts a word w, then A has a $\sqsubseteq$-smallest accepting run for w.

We use the $\sqsubseteq$-relation on run prefixes to order the vertices of V_i that belong to Q_d: for two different vertices $v = (q,i) \in V_i$ and $v' = (q',i) \in V_i$, v is $\sqsubset_i$-smaller than v', if there is a run prefix of G_w that ends up in v which is $\sqsubseteq$-smaller than all the run prefixes that ends up in v', which induces a total order among the vertices of V_i^d because the states in Q_d are totally ordered by the function Ord.

Lemma 1. *For all $i \geq 0$, for two different vertices $v = (q,i), v' = (q',i) \in V_i^d$, then either $v \sqsubset_i v'$ or $v' \sqsubset_i v$, i.e., $\sqsubset_i$ is a total order on V_i^d.*

Indexing vertices The index of a vertex $v = (q,i) \in V_i$ such that $q \in Q_d$, denoted by $\mathsf{Ind}_i(v)$, is a value in $\{1, 2, \dots, |Q_d|\}$ that denotes its order in V_i^d according to $\sqsubset_i$ (the $\sqsubset_i$-smallest element has index 1). For $i \geq 0$, we identify two important sets of vertices:

– $\mathsf{Dec}(V_i^d)$ is the set of vertices $v \in V_i^d$ such that there exists a vertex $v' \in V_{i+1}^d$: $(v, v') \in E$ and $\mathsf{Ind}_{i+1}(v') < \mathsf{Ind}_i(v)$, i.e. the set of vertices in V_i^d whose (unique) successor in V_{i+1}^d has a smaller index value.
– $\mathsf{Acc}(V_i^d)$ is the set of vertices $v = (q,i) \in V_i^d$ such that there exists $v' = (q', i+1) \in V_{i+1}^d$: $(v, v') \in E$ and $(q, w(i), q') \in \alpha$, i.e. the set of vertices in V_i^d that are the source of an accepting transition on $w(i)$.

Remark 2. Along a run, the index of vertices can only decrease. As the function $\mathsf{Ind}(\cdot)$ has a finite range, the index along a run has to eventually stabilize.

Assigning colors The set of colors that are used for coloring the levels of the run DAG G_w is $\{1, 2, \dots, 2 \cdot |Q_d| + 1\}$. We associate a color with each transition from level i to level $i+1$ according to the following set of cases:

1. if $\mathsf{Dec}(V_i^d) = \emptyset$ and $\mathsf{Acc}(V_i^d) \neq \emptyset$, the color is $2 \cdot \min_{v \in \mathsf{Acc}(V_i^d)} \mathsf{Ind}_i(v)$.
2. if $\mathsf{Dec}(V_i^d) \neq \emptyset$ and $\mathsf{Acc}(V_i^d) = \emptyset$, the color is $2 \cdot \min_{v \in \mathsf{Dec}(V_i^d)} \mathsf{Ind}_i(v) - 1$.

3. if $\mathsf{Dec}(V_i^d) \neq \emptyset$ and $\mathsf{Acc}(V_i^d) \neq \emptyset$, the color is defined as the minimal color among
   - $c_{\mathsf{odd}} = 2 \cdot \min_{v \in \mathsf{Dec}(V_i^d)} \mathsf{Ind}_i(v) - 1$, and
   - $c_{\mathsf{even}} = 2 \cdot \min_{v \in \mathsf{Acc}(V_i^d)} \mathsf{Ind}_i(v)$.
4. if $\mathsf{Dec}(V_i^d) = \mathsf{Acc}(V_i^d) = \emptyset$, the color is $2 \cdot |Q_q| + 1$.

The intuition behind this coloring is as follows: the coloring tracks runs in Q_d (only those are potentially accepting as $\alpha \subseteq Q_d \times \Sigma \times Q_d$) and tries to produce an even color that corresponds to the smallest index of an accepting run. If in level i the run DAG has an outgoing transition that is accepting, then this is a *positive event*, as a consequence the color emitted is *even* and it is a function of the smallest index of a vertex associated with an accepting transition from V_i to V_{i+1}. Runs in Q_d are deterministic but they can merge with *smaller* runs. When this happens, this is considered as a *negative event* because the even colors that have been emitted by the run that merges with the smaller run should not be taken into account anymore. As a consequence an odd color is emitted in order to cancel all the (good) even colors that were generated by the run that merges with the smaller one. In that case the odd color is function of the smallest index of a run vertex in V_i whose run merges with a smaller vertex in V_{i+1}. Those two first cases are handled by cases 1 and 2 of the case study above. When both situations happen at the same time, then the color is determined by the minimum of the two colors assigned to the positive and the negative events. This is handled by case 3 above. And finally, when there is no accepting transition from V_i to V_{i+1} and no merging, the largest odd color is emitted as indicated by case 4 above.

According to this intuition, we define the *color summary* of the run DAG G_w as the minimal color that appears infinitely often along the transitions between its levels. Because of the deterministic behavior of the automaton in Q_d, each run can only merge at most $|Q_d| - 1$ times with a smaller one (the size of the range of the function $\mathsf{Ind}(\cdot)$ minus one), and as a consequence of the definition of the above coloring, we know that, on word accepted by A, the smallest accepting run will eventually generate infinitely many (good) even colors that are never trumped by smaller odd colors.

Example 1. The left part of Fig. 2 depicts the run DAG of the limit-deterministic automaton of Fig. 1 on the word $w = abb(ab)^\omega$. Each path in this graph represents a run of the automaton on this word. The coloring of the run DAG follows the coloring rules defined above. Between level 0 and level 1, the color is equal to $7 = 2|Q_d| + 1$, as no accepting edge is taken from level 0 to level 1 and no run merges (within Q_d). The color 7 is also emitted from level 1 to level 2 for the same reason. The color 4 is emitted from level 2 to level 3 because the accepting edge $(3, b, 3)$ is taken and the index of state 3 in level 2 is equal to 2 (state 4 has index 1 as it is the end point of the smallest run prefix within Q_d). The color 3 is emitted from level 3 to level 4 because the run that goes from 3 to 4 merges with the smaller run that goes from 4 to 4. In order to cancel the even colors emitted by the run that goes from 3 to 4, color 3 is emitted. It cancels the even color 4 emitted before by this run. Afterwards, colors 3 is emitted forever. The color summary is 3 showing that there is no accepting run in the run DAG.

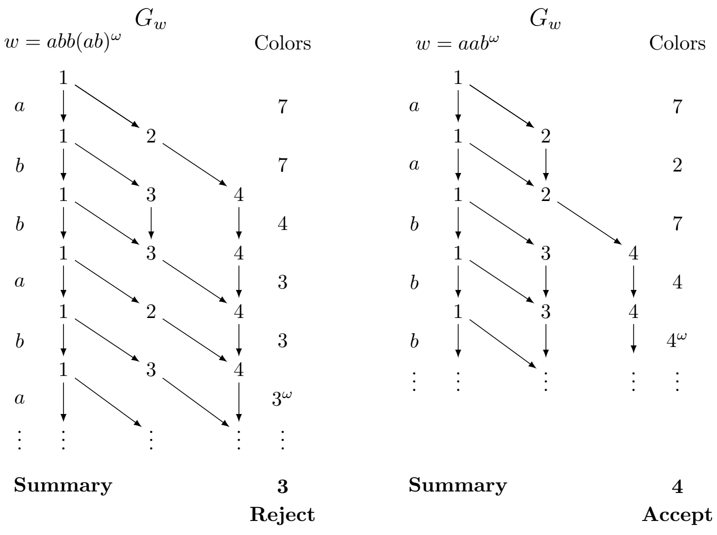


Fig. 2: The run DAGs automaton of Fig. 1 on the word $w = (ab)^\omega$ given on the left, and on the word $w = aab^\omega$ given on the right, together with their colorings.

The right part of Fig. 2 depicts the run DAG of the limit deterministic automaton of Fig. 1 on the word $w = aab^\omega$. The coloring of the run DAG follows the coloring rules defined above. Between levels 0 and 1, color 7 is emitted because no accepting edge is crossed. To the next level, we see the accepting edge $(2, a, 2)$ and color $2 \cdot 1 = 2$ is emitted. Upon reading the first b, we see again 7 since there is neither any accepting edge seen nor any merging takes place. Afterwards, each b causes an accepting edge $(3, b, 3)$ to be taken. While the smallest run, which visits 4 forever, is not accepting, the second smallest run that visits 3 forever is accepting. As 3 has index 2 in all the levels below level 3, the color is forever equal to 4. The color summary of the run is thus equal to $2 \cdot 2 = 4$ and this shows that word $w = aab^\omega$ is accepted by our limit deterministic automaton of Fig. 1.

The following theorem tells us that the color summary (the minimal color that appears infinitely often) can be used to identify run DAGs that contain accepting runs. The proof can be found in [EKRS17, Appendix A].

Theorem 1. *The color summary of the run DAG G_w is even if and only if there is an accepting run in G_w.*

3.2 Construction of the DPA

From an LDBA $A = (Q, Q_d, q_0, \Sigma, \delta, \alpha)$ and an ordering function $\mathsf{Ord} : Q \to \{1, 2, \ldots, |Q_d|, +\infty\}$ compatible with Q_d, we construct a deterministic parity automaton $B = (Q^B, q_0^B, \Sigma, \delta^B, p)$ that, on a word w, constructs the levels of the run DAG G_w and the coloring of previous section. Theorem 1 tells us that such an automaton accepts the same language as A.

First, we need some notations. Given a finite set S, we note $\mathcal{P}(S)$ the set of its subsets, and $\mathcal{OP}(S)$ the set of its totally ordered subsets. So if $(s, <) \in \mathcal{OP}(S)$

then $s \subseteq S$ and $< \subseteq s \times s$ is a total strict order on s. For $e \in s$, we denote by $\mathsf{Ind}_{(s,<)}(e)$ the position of $e \in s$ among the elements in s for the total strict order $<$, with the convention that the index of the $<$-minimum element is equal to 1. The deterministic parity automaton $B = (Q^B, q_0^B, \Sigma, \delta^B, p)$ is defined as follows.

States and initial state The set of states is $Q^B = \mathcal{P}(\overline{Q_d}) \times \mathcal{OP}(Q_d)$, i.e. a state of B is a pair $(s, (t, <))$ where s is a set of states outside Q_d, and t is an ordered subset of Q_d. The ordering reflects the relative index of each state within t. The initial state is $q_0^B = (\{q_0\}, (\{\}, \{\}))$.

Transition function Let $(s_1, (t_1, <_1))$ be a state in Q^B, and $\sigma \in \Sigma$. Then $\delta^B((s_1, (t_1, <_1))) = (s_2, (t_2, <_2))$ where:

- $s_2 = \mathsf{post}^\sigma_\delta(s_1) \cap \overline{Q_d}$;
- $t_2 = \mathsf{post}^\sigma_\delta(s_1 \cup t_1) \cap Q_d$;
- $<_2$ is defined from $<_1$ and Ord as follows: $\forall q_1, q_2 \in t_2$: $q_1 <_2 q_2$ iff:
  1. **either**, $\neg\exists q_1' \in t_1 : q_1 = \delta(q_1', \sigma)$, and $\neg\exists q_2' \in t_1 : q_2 = \delta(q_2', \sigma)$, and $\mathsf{Ord}(q_1) < \mathsf{Ord}(q_2)$,
     i.e. none has a predecessor in Q_d, then they are ordered using Ord;
  2. **or**, $\exists q_1' \in t_1 : q_1 = \delta(q_1', \sigma)$, and $\neg\exists q_2' \in t_1 : q_2 = \delta(q_2', \sigma)$,
     i.e. q_1 has a σ-predecessor in Q_d, and q_2 not;
  3. **or** $\exists q_1' \in t_1 : q_1 = \delta(q_1', \sigma)$, and $\exists q_2' \in t_1 : q_2 = \delta(q_2', \sigma)$, and $\min_{<_1}\{q_1' \in t_1 \mid q_1 = \delta(q_1', \sigma)\} < \min_{<_1}\{q_2' \in t_1 \mid q_2 = \delta(q_2', \sigma)\}$,
     i.e. both have a predecessor in Q_d, and they are ordered according to the order of their minimal parents.

Coloring To define the coloring of edges in the deterministic automaton, we need to identify the states $q \in t_1$ in a transition $(s_1, (t_1, <_1)) \xrightarrow{\sigma} (s_2, (t_2, <_2))$ whose indices decrease when going from t_1 to t_2. Those are defined as follows:

$$\mathsf{Dec}(t_1) = \{q_1 \in t_1 \mid \mathsf{Ind}_{(t_2,<_2)}(\delta(q_1, \sigma)) < \mathsf{Ind}_{(t_1,<_1)}(q_1)\}.$$

Additionally, let $\mathsf{Acc}(t_1) = \{q \mid \exists q' \in t_2 : (q, \sigma, q') \in \alpha\}$ denote the subset of states in t_1 that are the source of an accepting transition.

We assign a color to each transition $(s_1, (t_1, <_1)) \to^\sigma (s_2, (t_2, <_2))$ as follows:

1. if $\mathsf{Dec}(t_1) = \emptyset$ and $\mathsf{Acc}(t_1) \neq \emptyset$, the color is $2 \cdot \min_{q \in \mathsf{Acc}(t_1)} \mathsf{Ind}_{(t_1,<_1)}(q)$.
2. if $\mathsf{Dec}(t_1) \neq \emptyset$ and $\mathsf{Acc}(t_1) = \emptyset$, the color is $2 \cdot \min_{q \in \mathsf{Dec}(t_1)} \mathsf{Ind}_{(t_1,<_1)}(q) - 1$.
3. if $\mathsf{Dec}(t_1) \neq \emptyset$ and $\mathsf{Acc}(t_1) \neq \emptyset$, the color is defined as the minimal color among
   - $c_{\mathsf{odd}} = 2 \cdot \min_{q \in \mathsf{Dec}(t_1)} \mathsf{Ind}_{(t_1,<_1)}(q) - 1$, and
   - $c_{\mathsf{even}} = 2 \cdot \min_{q \in \mathsf{Acc}(t_1)} \mathsf{Ind}_{(t_1,<_1)}(q)$.
4. if $\mathsf{Dec}(t_1) = \mathsf{Acc}(t_1) = \emptyset$, the color is $2 \cdot |Q_q| + 1$.

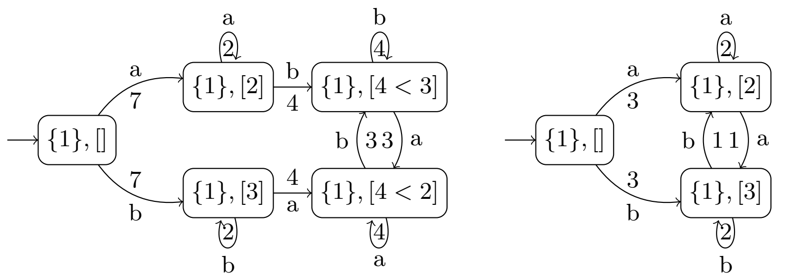


Fig. 3: Left: DPA that accepts the LTL language $\mathbf{FG}a \vee \mathbf{FG}b$, edges are decorated with a natural number that specifies its color. Right: A reduced DPA.

Example 2. The DPA of Fig. 3 is the automaton that is obtained by applying the construction LDBA→DPA defined above to the LDBA of Fig. 1 that recognizes the LTL language $\mathbf{FG}a \vee \mathbf{FG}b$. The figure only shows the reachable states of this construction. As specified in the construction above, states of DPA are labelled with a subset of $\overline{Q_d}$ and a ordered subset of Q_d of the original NBA. As an illustration of the definitions above, let us explain the color of edges from state $(\{1\}, [4, 3])$ to itself on letter b. When the NBA is in state 1, 3 or 4 and letter b is read, then the next state of the automaton is again 1, 3 or 4. Note also that there are no runs that are merging in that case. As a consequence, the color that is emitted is even and equal to the index of the smallest state that is the target of an accepting transition. In this case, this is state 3 and its index is 2. This is the justification for the color 4 on the edge. On the other hand, if letter a is read from state $(\{1\}, [4, 3])$, then the automaton moves to states $(\{1\}, [4, 2])$. The state 3 is mapped to state 4 and there is a run merging which induces that the color emitted is odd and equal to 3. This 3 trumps all the 4's that were possibly emitted from state $(\{1\}, [4, 3])$ before.

Theorem 2. *The language defined by the deterministic parity automaton B is equal to the language defined by the limit deterministic automaton A, i.e.* $\mathsf{L}(A) = \mathsf{L}(B)$.

Proof. Let $w \in \Sigma^\omega$ and G_w be the run DAG of A on w. It is easy to show by induction that the sequence of colors that occur along G_w is equal to the sequence of colors defined by the run of the automaton B on w. By Theorem 1, the language of automaton B is thus equal to the language of automaton A. □

3.3 Complexity Analysis

Upper bound Let $n = |Q|$ be the size of the LDBA and let $n_d = |Q_d|$ be the size of the accepting component. We can bound the number of different orderings using the series of reciprocals of factorials (with e being Euler's number):

$$|\mathcal{OP}(Q_d)| = \sum_{i=0}^{n_d} \frac{n_d!}{(n_d - i)!} \leq n_d \cdot n_d! \cdot \sum_{i=0}^{\infty} \frac{1}{i!} = e \cdot n_d \cdot n_d! \in \mathcal{O}(2^{n \cdot \log n})$$

Thus the obtained DPA has $\mathcal{O}(2^n \cdot 2^{n \cdot \log n}) = 2^{\mathcal{O}(n \cdot \log n)}$ states and $\mathcal{O}(n)$ colours.

Lower bound We obtain a matching lower bound by strengthening Theorem 8 from [Löd99]:

Lemma 2. *There exists a family $(L_n)_{n \geq 2}$ of languages (L_n over an alphabet of n letters) such that for every n the language L_n can be recognized by a limit-deterministic Büchi automaton with $3n+2$ states but can not be recognized by a deterministic Parity automaton with less than $n!$ states.*

Proof. The proof of Theorem 8 from [Löd99] constructs a non-deterministic Büchi automaton of exactly this size and which is in fact limit-deterministic.

Assume there exists a deterministic Parity automata for L_n with $m < n!$ states. Since parity automata are closed under complementation, we can obtain a parity automaton and hence also a Rabin automaton of size m for $\overline{L_n}$ and thus a Streett automaton of size m for L_n, a contradiction to Theorem 8 of [Löd99]. $\square$

Corollary 1. *Every translation from limit-deterministic Büchi automata of size n to deterministic parity yields automata with $2^{\Omega(n \log n)}$ states in the worst case.*

4 From LTL to Parity in $2^{2^{\mathcal{O}(n)}}$

In [SEJK16] we present a LTL→LDBA translation. Given a formula φ of size n, the translation produces an asymptotically optimal LDBA with $2^{2^{\mathcal{O}(n)}}$ states. The straightforward composition of this translation with the single exponential LDBA→DPA translation of the previous section is only guaranteed to be triple exponential, while the Safra-Piterman construction produces a DPA of at most doubly exponential size. In this section we describe a modified composition that yields a double exponential DPA. To the best of our knowledge this is is the first translation of the whole LTL to deterministic parity automata that is asymptotically optimal and does not use Safra's construction.

The section is divided into two parts. In the first part, we explain and illustrate a redundancy occurring in our LDBA→DPA translation, responsible for the undesired extra exponential. We also describe an optimization that removes this redundancy when the LDBA satisfies some conditions. In the second part, we show these conditions are satisfied on the products of the LTL→LDBA translation, which in turn guarantees a doubly exponential LTL→DPA procedure.

4.1 An improved construction

We can view the second component of a state of the DPA as a sequence of states of the LDBA, ordered by their indices. Since there are $2^{2^{\mathcal{O}(n)}}$ states of the LDBA for an LTL formula of length n, the number of such sequences is

$$2^{2^{\mathcal{O}(n)}}! = 2^{2^{2^{\mathcal{O}(n)}}}$$

If only the length of the sequences (the maximum index) were bounded by 2^n, the number of such sequences would be smaller than the number of functions $2^n \to 2^{2^{\mathcal{O}(n)}}$ which is

$$(2^{2^{\mathcal{O}(n)}})^{2^n} = 2^{2^{\mathcal{O}(n)} \cdot 2^n} = 2^{2^{\mathcal{O}(n)}}$$

Fix an LDBA with set of states Q. Assume the existence of an *oracle*: a list of statements of the form $\mathsf{L}(q) \subseteq \bigcup_{q' \in Q_q} \mathsf{L}(q')$ where $q \in Q$ and $Q_q \subseteq Q$. We use the oracle to define a mapping that associates to each run DAG G_w a "reduced DAG" G_w^*, defined as the result of iteratively performing the following four-step operation:

– Find the first V_i in the current DAG such that the sequence $(v_1, i) \sqsubset (v_2, i) \sqsubset \cdots \sqsubset (v_{n_i}, i)$ of vertices of V_i^d contains a vertex (v_k, i) for which the oracle ensures
$$\mathsf{L}(v_k) \subseteq \bigcup_{j<k} \mathsf{L}(v_j) \qquad (*)$$
We call (v_k, i) a *redundant vertex.*
– Remove (v_k, i) from the sequence, and otherwise keep the ordering $\sqsubseteq_i$ unchanged (thus decreasing the index of vertices (v, ℓ) with $\ell > k$).
– Redirect transitions leading from vertices in V_{i-1} to (v_k, i) so that they lead to the smallest vertex (v_1, i) of V_i.
– Remove any vertices (if any) that are no longer reachable from vertices of V_1.

We define the color summary of G_w^* in exactly the same way as the color summary of G_w. The DAG G_w^* satisfies the following crucial property, whose proof can be found in [EKRS17, Appendix B]:

Proposition 1. *The color summary of the run DAG G_w^* is even if and only if there is an accepting run in G_w.*

The mapping on DAGs induces a reduced DPA as follows. The states are the pairs $(s, (t, <))$ such that $(t, <)$ does not contain redundant vertices. There is a transition $(s_1, (t_1, <)) \xrightarrow{a} (s_2, (t_2, <))$ with color c iff there is a word w and an index i such that $(s_1, (t_1, <))$ and $(s_2, (t_2, <))$ correspond to the i-th and $(i+1)$-th levels of G_w^*, and a and c are the letter and color of the step between these levels in G_w^*. Observe that the set of transitions is independent of the words chosen to define them.

The equivalence between the initial DPA $\mathcal{A}$ and the reduced DPA $\mathcal{A}_r$ follows immediately from Proposition 1: $\mathcal{A}$ accepts w iff G_w contains an accepting run iff the color summary of G_w^* is even iff $\mathcal{A}_r$ accepts w.

Example 3. Consider the LDBA of Fig. 1 and an oracle given by $\mathsf{L}(4) = \emptyset$, ensuring $\mathsf{L}(4) \subseteq \bigcup_{i \in I} \mathsf{L}(i)$ for any $I \subseteq Q$. Then 4 is always redundant and merged, removing the two rightmost states of the DPA of Fig. 3 (left), resulting in the DPA of Fig. 3 (right). However, for the sake of technical convenience, we shall refrain from removing a redundant vertex when it is the smallest one (with index 1).

Since the construction of the reduced DPA is parametrized by an oracle, the obvious question is how to obtain an oracle that does not involve applying an expensive language inclusion test. Let us give a first example in which an oracle can be easily obtained:

Example 4. Consider an LDBA where each state $v = \{s_1, \dots, s_k\}$ arose from some powerset construction on an NBA in such a way that $\mathsf{L}(\{s_1, \dots, s_k\}) = \mathsf{L}(s_1) \cup \cdots \mathsf{L}(s_k)$. An oracle can, for instance, allow us to merge whenever $v_k \subseteq \bigcup_{j<k} v_j$, which is a sound syntactic approximation of language inclusion. This motivates the following formal generalization.

Let $\mathcal{L}_B = \{L_i \mid i \in B\}$ be a finite set of languages, called *base* languages. We call $\mathcal{L}_C := \{\bigcup \mathcal{L} \mid \mathcal{L} \subseteq \mathcal{L}_B\}$ the join-semilattice of *composed* languages. We shall assume an LDBA with some $\mathcal{L}_B$ such that $\mathsf{L}(q) \in \mathcal{L}_C$ for every state q. We say that such an LDBA *has a base* $\mathcal{L}_B$. In other words, every state recognizes a union of some base languages. (Note that every automaton has a base of at most linear size.) Whenever we have states v_j recognizing $\bigcup_{i \in I_j} L_i$ with $I_j \subseteq B$ for every j, the oracle allows us to merge vertices v_k satisfying $I_k \subseteq \bigcup_{j<k} I_j$. Intuitively, the oracle declares a vertex redundant whenever the simple syntactic check on the indices allows for that.

Let $V_1 = \bigcup_{i \in I_1} L_i, \cdots V_j = \bigcup_{i \in I_j} L_i$ be a sequence of languages of $\mathcal{L}_C$ where the reduction has been applied and there are no more redundant vertices. The maximum length of such a sequence is given already by the base $\mathcal{L}_B$ and we denote it $\mathit{width}(\mathcal{L}_B)$.

Lemma 3. *For any $\mathcal{L}_B$, we have $\mathit{width}(\mathcal{L}_B) \leq |\mathcal{L}_B| + 1$.*

Proof. We provide an injective mapping of languages in the sequence (except for V_1) into B. Since $I_2 \not\subseteq I_1$, there is some $i \in I_2 \setminus I_1$ and we map V_2 to this i. In general, since $I_k \not\subseteq \bigcup_{j=1}^{k-1} I_j$, we also have $i \in I_k \setminus \bigcup_{j=1}^{k-1} I_j$ and we map V_k to this i. $\square$

On the one hand, the transformation of LDBA to DPA without the reduction yields $2^{\mathcal{O}(|Q| \cdot \log |Q|)}$ states. On the other hand, we can now show that the second component of reduced LDBA with a base can be exponentially smaller. Further, let us assume the LDBA is *initial-deterministic*, meaning that $\delta \cap (\overline{Q_d} \times \Sigma \times \overline{Q_d})$ is deterministic, thus not resulting in blowup in the first component.

Corollary 2. *For every initial-deterministic LDBA with base of size m, there is an equivalent DPA with $2^{\mathcal{O}(m^2)}$ states.*

Proof. The number of composed languages is $\mathcal{L}_C = 2^m$. Therefore, the LDBA has at most 2^m (non-equivalent) states. Hence the construction produces at most

$$|\mathcal{L}_C| \cdot |\mathcal{L}_C|^{\mathcal{O}(\mathit{width}(\mathcal{L}_B))} = 2^m \cdot (2^m)^{\mathcal{O}(m)} = 2^{\mathcal{O}(m^2)}$$

states since the LDBA is initial-deterministic, causing no blowup in the first component. $\square$

4.2 Bases for LDBAs Obtained from LTL Formulas

We prove that the width for LDBA arising from the LTL transformation is only singly exponential in the formula size. To this end, we need to recall a property of the LTL$\rightarrow$LDBA translation of [SEJK16]. Since partial evaluation of formulas plays a major role in the translation, we introduce the following definition. Given an LTL formula φ and sets T and F of LTL formulas, let $\varphi[T, F]$ denote the result of substituting **tt** (true) for each occurrence of a formula of T in φ, and similarly **ff** (false) for formulas of F. The following property of the translation is proven in [EKRS17, Appendix C].

Proposition 2. *For every LTL formula φ, every state s of the LDBA of [SEJK16] is labelled by an LTL formula $label(s)$ such that (i) $\mathsf{L}(s) = \mathsf{L}(label(s))$ and (ii) $label(s)$ is a Boolean combination of subformulas of $\varphi[T_s, F_s]$ for some T_s and F_s. Moreover, the LDBA is initial-deterministic.*

As a consequence, we can bound the corresponding base:

Corollary 3. *For every LTL formula φ, the LDBA of [SEJK16] for φ has a base of size $2^{\mathcal{O}(|\varphi|)}$.*

Proof. Firstly, we focus on states using the same $\varphi[T_s, F_s]$. The language of each state can be defined by a Boolean formula over $\mathcal{O}(|\varphi|)$ atoms. Since every Boolean formula can be expressed in the disjunctive normal form, its language is a union of the conjuncts. The conjunctions thus form a base for these states. There are exponentially many different conjunction in the number of atoms. Hence the base is of singly exponential size $2^{\mathcal{O}(|\varphi|)}$ as well.

Secondly, observe that there are only $2^{\mathcal{O}(|\varphi|)}$ different formulas $\varphi[T_s, F_s]$ and thus only $2^{\mathcal{O}(|\varphi|)}$ different sets of atoms. Altogether, the size is bounded by

$$2^{\mathcal{O}(|\varphi|)} \cdot 2^{\mathcal{O}(|\varphi|)} = 2^{\mathcal{O}(|\varphi|)} \qquad \square$$

Theorem 3. *For every LTL formula φ, there is a DPA with $2^{2^{\mathcal{O}(|\varphi|)}}$ states.*

Proof. The LDBA for φ has base of singly exponential size $2^{\mathcal{O}(|\varphi|)}$ by Corollary 3 and is initial-deterministic by Proposition 2. Therefore, by Corollary 2, the size of the DPA is doubly exponential, in fact

$$2^{(2^{\mathcal{O}(|\varphi|)})^2} = 2^{2^{\mathcal{O}(|\varphi|)}} \qquad \square$$

This matches the lower bound $2^{2^{\Omega(n)}}$ by [KR10] as well as the upper bound by the Safra-Piterman approach. Finally, note that while the breakpoint constructions in [SEJK16] is analogous to Safra's vertical merging, the merging introduced here is analogous to Safra's horizontal merging.

5 Experimental Evaluation

We evaluate the performance of our construction on several datasets taken from [BKS13,DWDMR08,SEJK16] and several Temporal Logic Synthesis Format (TLSF) specifications [JBB$^+$16] of the SyntComp 2016 competition.

We use the size of the constructed deterministic automaton as an indicator for the overall performance of the synthesis procedure. In [ST03] it is argued that the degree of determinism of the automaton is a better predictor for performance in model-checking problems; however, this parameter is not applicable for synthesis problems, which require deterministic automata.

We compare two versions of our implementation (with and without optimizations, see below) with the algorithms of Spot [DLLF$^+$16]. Each tool is given 64GB of memory and 10 minutes. Increasing time to 10 hours does not change the results. More precisely, we compare the following three setups:

S. (`ltl2tgba`, 2.1.1) - Spot [DLLF$^+$16] implements a version of the Safra-Piterman determinization procedure [Red12] with several optimizations.

L2P and L2P′. (`ltl2dpa`, 1.0.0) - L2P is the construction of this paper, available at `www7.in.tum.de/~sickert/projects/ltl2dpa`. L2P′ adds two optimizations. First, the tool translates both the formula and its negation to DPAs A_1, A_2, complements A_2 to yield $\overline{A}_2$, and picks the smaller of A_1, A_2. Further, we apply the simplification routines of Spot (`ltlfilt` and `autfilt`, respectively).

We consider three groups of benachmarks:

Parametric Formulas. 10 benchmarks from [BKS13,SEJK16]). In six cases S and L2P′ produce identical results. The other four are

$$\begin{array}{ll} R(n) = \bigwedge_{i=1}^{n}(\mathbf{GF}p_i \vee \mathbf{FG}p_{i+1}) & G(n) = (\bigwedge_{i=1}^{n}\mathbf{GF}p_i) \rightarrow (\bigwedge_{i=1}^{n}\mathbf{GF}q_i) \\ \theta(n) = \neg((\bigwedge_{i=1}^{n}\mathbf{GF}p_i) \rightarrow \mathbf{G}(q \rightarrow \mathbf{F}r)) & F(n) = \bigwedge_{i=1}^{n}(\mathbf{GF}p_i \rightarrow \mathbf{GF}q_i) \end{array}$$

for which the results are shown in (figure 4a). Additionally, we consider the "f" formulas from [SEJK16] (table 1). Observe that L2P′ performs clearly better, and the gap between the tools grows when the parameter increases.

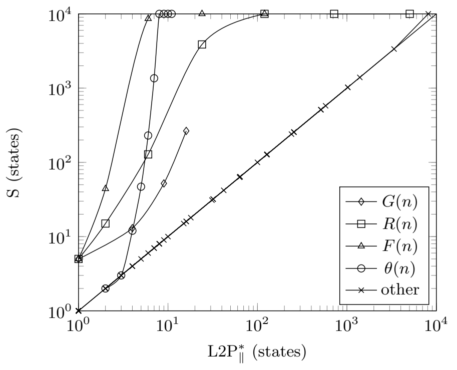
Randomly Generated Formulas from [BKS13] (figure 4b).

Real Data. Formulas taken from case studies and synthesis competitions — the intended domain of application of our approach. Figures 4c and 4d show results for the real-world formulas of [BKS13] and the TLSF specifications contained in the Acacia set of [JBB$^+$16]. Table 1 shows results for LTL formulas expressing properties of Szymanski's protocol [DWDMR08], and for the generalised buffer benchmark of Acacia.

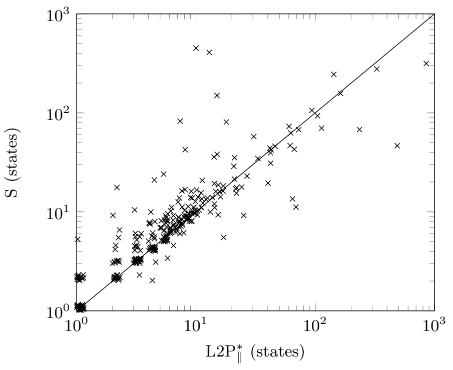
Average Compression Ratios. The geometric average compression ratio for a benchmark suite B is defined as $\prod_{\varphi \in B}(n_\varphi^S / n_\varphi^{L2P'})^{1/|B|}$, where n_φ^S and $n_\varphi^{L2P'}$ denote the number of states of the automata produced by Spot and L2P′, respectively. The ratios in our experiments (excluding benchmarks where Spot times out) are: 1.14 for random formulas, 1.12 for the real-world formulas of [BKS13], and 1.35 for the formulas of Acacia.

6 Conclusion

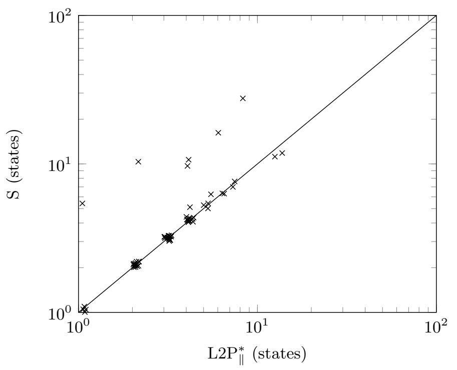
We have presented a simple, "Safraless", and asymptotically optimal translation from LTL and LDBA to deterministic parity automata. Furthermore, the



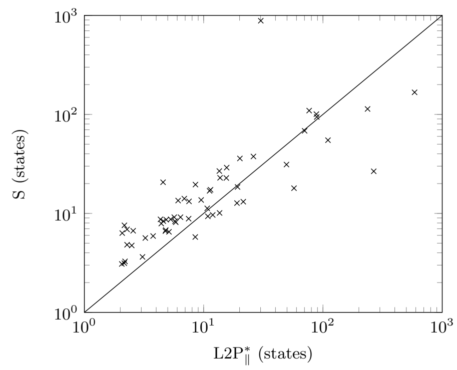
(a) Parametrised Formulas



(b) Randomly Generated Formulas



(c) "Real-world" Formulas



(d) TLSF/Acacia

Fig. 4: Comparison of Spot and our implementation using the best configurations. Timeouts are denoted by setting the size of the automaton to the maximum.

Table 1: Number of states and number of used colours in parenthesis for the constructed automata. Timeouts are marked with t.

	$f(1,0)$	$f(1,2)$	$f(1,4)$	$f(2,0)$	$f(2,2)$	zn	zp1	zp2	zp3	Buffer
S	18(6)	141(8)	2062(8)	208(12)	883(12)	t	t	t	t	t
L2P	12(8)	114(9)	332(15)	144(14)	4732(19)	t	t	t	t	1425(27)
L2P′	12(8)	78(7)	271(11)	106(9)	1904(15)	32(6)	42(6)	111(12)	97(12)	435(4)

translation is suitable for an on-the-fly implementation. The resulting automata are substantially smaller than those produced by the SPOT library for formulas obtained from synthesis specifications, and have comparable or smaller size for other benchmarks. In future work we want to investigate the performance of the translation as part of a synthesis toolchain.

Acknowledgments. The authors want to thank Michael Luttenberger for helpful discussions and the anonymous reviewers for constructive feedback.

References

BHS$^+$16. Frantisek Blahoudek, Matthias Heizmann, Sven Schewe, Jan Strejcek, and Ming-Hsien Tsai. Complementing semi-deterministic Büchi automata. In *TACAS*, volume 9636 of *LNCS*, pages 770–787, 2016.

BKS13. Frantisek Blahoudek, Mojmír Křetínský, and Jan Strejček. Comparison of LTL to deterministic Rabin automata translators. In *LPAR*, volume 8312 of *LNCS*, pages 164–172, 2013.

CY95. Costas Courcoubetis and Mihalis Yannakakis. The complexity of probabilistic verification. *J. ACM*, 42(4):857–907, 1995.

DLLF$^+$16. Alexandre Duret-Lutz, Alexandre Lewkowicz, Amaury Fauchille, Thibaud Michaud, Etienne Renault, and Laurent Xu. Spot 2.0 — a framework for LTL and ω-automata manipulation. In *Proceedings of the 14th International Symposium on Automated Technology for Verification and Analysis (ATVA'16)*, volume 9938 of *Lecture Notes in Computer Science*, pages 122–129. Springer, October 2016. To appear.

DWDMR08. Martin De Wulf, Laurent Doyen, Nicolas Maquet, and Jean-François Raskin. Antichains - Alternative Algorithms for LTL Satisfiability and Model-Checking. *TACAS*, 2008.

EKRS17. Javier Esparza, Jan Křetínský, Jean-François Raskin, and Salomon Sickert. From LTL and limit-deterministic büchi automata to deterministic parity automata. Technical Report abs/1701.06103, arXiv.org, 2017.

Fin15. Bernd Finkbeiner. Automata, games, and verification, 2015. Available at `https://www.react.uni-saarland.de/teaching/automata-games-verification-15/downloads/notes.pdf`.

FKVW15. Seth Fogarty, Orna Kupferman, Moshe Y. Vardi, and Thomas Wilke. Profile trees for büchi word automata, with application to determinization. *Inf. Comput.*, 245:136–151, 2015.

JBB$^+$16. Swen Jacobs, Roderick Bloem, Romain Brenguier, Ayrat Khalimov, Felix Klein, Robert Könighofer, Jens Kreber, Alexander Legg, Nina Narodytska, Guillermo A. Pérez, Jean-François Raskin, Leonid Ryzhyk, Ocan Sankur, Martina Seidl, Leander Tentrup, and Adam Walker. The 3rd reactive synthesis competition (SYNTCOMP 2016): Benchmarks, participants & results. *CoRR*, abs/1609.00507, 2016.

KR10. Orna Kupferman and Adin Rosenberg. The blowup in translating LTL to deterministic automata. In *MoChArt*, volume 6572 of *LNCS*, pages 85–94. Springer, 2010.

KV01. Orna Kupferman and Moshe Y. Vardi. Weak alternating automata are not that weak. *ACM Trans. Comput. Log.*, 2(3):408–429, 2001.

KV15. Dileep Kini and Mahesh Viswanathan. Limit deterministic and probabilistic automata for LTL $\setminus$ GU. In *TACAS*, volume 9035 of *LNCS*, pages 628–642, 2015.

Löd99. Christof Löding. Optimal bounds for transformations of omega-automata. In C. Pandu Rangan, Venkatesh Raman, and Ramaswamy Ramanujam, editors, *Foundations of Software Technology and Theoretical Computer Science, 19th Conference, Chennai, India, December 13-15, 1999, Proceedings*, volume 1738 of *Lecture Notes in Computer Science*, pages 97–109. Springer, 1999.

Pit07. Nir Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. *Logical Methods in Computer Science*, 3(3), 2007.

Red12. Roman R. Redziejowski. An improved construction of deterministic omega-automaton using derivatives. *Fundam. Inform.*, 119(3-4):393–406, 2012.

Saf88. Shmuel Safra. On the complexity of omega-automata. In *FOCS*, pages 319–327, 1988.

SEJK16. Salomon Sickert, Javier Esparza, Stefan Jaax, and Jan Kretínský. Limit-deterministic büchi automata for linear temporal logic. In *Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part II*, pages 312–332, 2016.

ST03. Roberto Sebastiani and Stefano Tonetta. "more deterministic" vs. "smaller" büchi automata for efficient LTL model checking. In *Correct Hardware Design and Verification Methods, 12th IFIP WG 10.5 Advanced Research Working Conference, CHARME 2003, L'Aquila, Italy, October 21-24, 2003, Proceedings*, pages 126–140, 2003.

Var85. Moshe Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *FOCS*, pages 327–338, 1985.

VW86. Moshe Y. Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *LICS*, pages 332–344, 1986.

One Theorem to Rule Them All: A Unified Translation of LTL into ω-Automata[*]

Javier Esparza
esparza@in.tum.de
Technische Universität München
Germany

Jan Křetínský
jan.kretinsky@in.tum.de
Technische Universität München
Germany

Salomon Sickert
sickert@in.tum.de
Technische Universität München
Germany

Abstract

We present a unified translation of LTL formulas into deterministic Rabin automata, limit-deterministic Büchi automata, and nondeterministic Büchi automata. The translations yield automata of asymptotically optimal size (double or single exponential, respectively). All three translations are derived from one single Master Theorem of purely logical nature. The Master Theorem decomposes the language of a formula into a positive boolean combination of languages that can be translated into ω-automata by elementary means. In particular, Safra's, ranking, and breakpoint constructions used in other translations are not needed.

CCS Concepts • **Theory of computation** → **Automata over infinite objects**; **Modal and temporal logics**;

Keywords Linear temporal logic, Automata over infinite words, Deterministic automata, Non-deterministic automata

1 Introduction

Linear temporal logic (LTL) [32] is a prominent specification language, used both for model checking and automatic synthesis of systems. In the standard automata-theoretic approach [38] the input formula is first translated into an ω-automaton, and then the product of this automaton with the input system is further analyzed. Since the size of the product is often the bottleneck of all the verification algorithms, it is crucial that the ω-automaton is as small as possible. Consequently, a lot of effort has been spent on translating LTL into small automata, e.g. [4, 10–12, 17, 18, 20, 21, 36].

While non-deterministic Büchi automata (NBA) can be used for model checking non-deterministic systems, other applications such as model checking probabilistic systems or synthesis usually require automata with a certain degree of determinism, such as deterministic parity automata (DPA) or deterministic Rabin automata (DRA) [5], deterministic generalized Rabin automata (DGRA) [8], limit-deterministic (or semi-deterministic) Büchi automata (LDBA) [9, 22, 35, 37], unambiguous Büchi automata [6] etc. The usual constructions that produce such automata are based on Safra's determinization and its variants [31, 33, 34]. However, they are known to be difficult to implement efficiently, and to be practically inefficient in many cases due to their generality. Therefore, a recent line of work shows how DPA [14, 28], DRA and DGRA [13, 15, 26, 27], or LDBA [23, 24, 35] can be produced directly from LTL, without the intermediate step through a non-deterministic automaton. All these works share the principle of describing each state by a collection of formulas, as happens in the classical tableaux construction for translation of LTL into NBA. This makes the approach particularly apt for semantic-based state reductions, e.g., for merging states corresponding to equivalent formulas. These reductions cannot be applied to Safra-based constructions, where this semantic structure gets lost.

In this paper, we provide a unified view of translations of LTL into NBA, LDBA, and DRA enjoying the following properties, absent in former translations:

Asymptotic Optimality. D(G)RA are the most compact among the deterministic automata used in practice, in particular compared to DPA. Previous translations to D(G)RA were either limited to fragments of LTL [3, 26, 27], or only shown to be triply exponential [13, 15]. Here we provide constructions for all mentioned types of automata matching the optimal double exponential bound for DRA and LDBA, and the optimal single exponential bound for NBA.

Symmetry. The first translations [26, 27] used auxiliary automata to monitor each *Future*- and *Globally*-subformula. While this approach worked for fragments of LTL, subsequent constructions for full LTL [13, 15, 35] could not preserve the symmetric treatment. They only used auxiliary automata for **G**-subformulas, at the price of more complex constructions. Our translation re-establishes the symmetry of the first constructions. It treats **F** and **G** equally (actually, and more generally, it treats each operator and its dual equally), which results into simpler auxiliary automata.

Independence of Syntax. Previous translations were quite sensitive to the operators used in the syntax of LTL. In particular, the only greatest-fixed-point operator they allowed was *Globally*. Since formulas also had to be in negation normal form, pre-processing of the input often led to unnecessarily large formulas. While our translations still requires negation normal form, it allows for direct treatment of *Release*, *Weak until*, and other operators.

Unified View. Our translations rely on a novel *Master Theorem*, which decomposes the language of a formula into a positive boolean combination of "simple" languages, in the sense that they are easy to translate into automata. This approach is arguably simpler than previous ones (it is certainly simpler than our previous papers [15, 35]). Besides, it provides a unified treatment of DRA, NBA, and LDBA, differing only in the translations of the "simple" languages. The automaton for the formula is obtained from the automata for the "simple" languages by means of standard operations for closure under union and intersection.

[*]This work was partially funded and supported by the Czech Science Foundation, grant No. P202/12/G061, and the German Research Foundation (DFG) project "Verified Model Checkers" (317422601).

LICS '18, July 9–12, 2018, Oxford, United Kingdom
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5583-4/18/07…$15.00
https://doi.org/10.1145/3209108.3209161

On top of its theoretical advantages, our translation is comparable to previous DRA translations in practice, even without major optimizations. Summarizing, we think this paper finally achieves the goals formulated in [26], where the first translation of this kind—valid only for what we would now call a small fragment of LTL—was presented.

Structure of the Paper. Section 2 contains preliminaries about LTL and ω-automata. Section 3 introduces some definitions and results of [15, 35]. Section 4 shows how to use these notions to translate four simple fragments of LTL into deterministic Büchi and coBüchi automata; these translations are later used as building blocks. Section 5 presents our main result, the Master Theorem. Sections 6, 7, and 8 apply the Master Theorem to derive translations of LTL into DRA, NBA, and LDBA, respectively. Section 9 compares the paper to related work and puts the obtained results into context. The appendix of the accompanying technical report [16] contains the few omitted proofs and further related material.

2 Preliminaries

2.1 ω-Languages and ω-Automata

Let Σ be a finite alphabet. An ω-word w over Σ is an infinite sequence of letters $w[0]w[1]w[2]\dots$. We denote the finite infix $w[i]w[i{+}1]\cdots w[j{-}1]$ by w_{ij}, and the infinite suffix $w[i]w[i{+}1]\dots$ by w_i. An ω-language is a set of ω-words.

For the sake of presentation, we introduce ω-automata with accepting conditions defined on states. However, all results can be restated with accepting conditions defined on transitions, more in line with other recent papers and tools [2, 12, 25].

Let Σ be a finite alphabet. A *nondeterministic pre-automaton* over Σ is a tuple $\mathcal{P} = (Q, \Delta, Q_0)$ where Q is a finite set of states, $\Delta\colon Q \times \Sigma \to 2^Q$ is a transition function, and Q_0 is a set of initial states. A transition is a triple (q, a, q') such that $q' \in \Delta(q, a)$. A pre-automaton $\mathcal{P}$ is deterministic if Q_0 is a singleton and $\Delta(q, a)$ is a singleton for every $q \in Q$ and $a \in \Sigma$.

A *run* of $\mathcal{P}$ on an ω-word w is an infinite sequence of states $r = q_0q_1q_2\dots$ with $q_{i+1} \in \delta(q_i, w[i])$ for all i and we denote by $\mathit{inf}(r)$ the set of states occurring infinitely often in r. An *accepting condition* is an expression over the syntax $\alpha ::= \mathit{inf}(S) \mid \mathit{fin}(S) \mid \alpha_1 \vee \alpha_2 \mid \alpha_1 \wedge \alpha_2$ with $S \subseteq Q$. Accepting conditions are evaluated on runs and the evaluation relation $r \models \alpha$ is defined as follows:

$$\begin{array}{llll} r & \models & \mathit{inf}(S) & \text{iff} \quad \mathit{inf}(r) \cap S \neq \emptyset \\ r & \models & \mathit{fin}(S) & \text{iff} \quad \mathit{inf}(r) \cap S = \emptyset \\ r & \models & \alpha_1 \vee \alpha_2 & \text{iff} \quad r \models \alpha_1 \text{ or } r \models \alpha_2 \\ r & \models & \alpha_1 \wedge \alpha_2 & \text{iff} \quad r \models \alpha_1 \text{ and } r \models \alpha_2 \end{array}$$

An accepting condition α is a

- Büchi condition if $\alpha = \mathit{inf}(S)$ for some set S of states.
- coBüchi condition if $\alpha = \mathit{fin}(S)$ for some set S of states.
- Rabin condition if $\alpha = \bigvee_{i=1}^{k}(\mathit{inf}(I_i) \wedge \mathit{fin}(F_i))$ for some $k \geq 1$ and some sets $I_1, F_1, \dots, I_k, F_k$ of states.

An ω-automaton over Σ is a tuple $\mathcal{A} = (Q, \Delta, Q_0, \alpha)$ where (Q, Δ, Q_0) is a pre-automaton over Σ and α is an accepting condition. A run r of $\mathcal{A}$ is *accepting* if $r \models \alpha$. A word w is accepted by $\mathcal{A}$ if some run of $\mathcal{A}$ on w is accepting. An ω-automaton is a Büchi (coBüchi, Rabin) automaton if its accepting condition is a Büchi (coBüchi, Rabin) condition.

Limit-Deterministic Büchi Automata. Intuitively, a NBA is limit-deterministic if it can be split into a non-deterministic component without accepting states, and a deterministic component. The automaton can only accept by "jumping" from the non-deterministic to the deterministic component, but after the jump it must stay in the deterministic component forever. Formally, a NBA $\mathcal{B} = (Q, \Delta, Q_0, \alpha)$ is *limit-deterministic* (LDBA) if Q can be partitioned into two disjoint sets $Q = Q_{\mathcal{N}} \uplus Q_{\mathcal{D}}$, s.t.

1. $\Delta(q, \nu) \subseteq Q_{\mathcal{D}}$ and $|\Delta(q, \nu)| = 1$ for every $q \in Q_{\mathcal{D}}$, $\nu \in \Sigma$, and
2. $S \subseteq Q_{\mathcal{D}}$ for all $S \in \alpha$.

2.2 Linear Temporal Logic

We work with a syntax for LTL in which formulas are written in negation-normal form, i.e., negations only occur in front of atomic propositions. For every temporal operator we also include in the syntax its dual operator. On top of the next operator **X**, which is self-dual, we introduce temporal operators **F** (eventually), **U** (until), and **W** (weak until), and their duals **G** (always), **R** (release) and **M** (strong release). The syntax may look redundant but as we shall see it is essential to include **W** and **M** and very convenient to include **F** and **G**.

Syntax and semantics of LTL. A formula of LTL in *negation normal form* over a set of atomic propositions (Ap) is given by the syntax:

$$\begin{aligned} \varphi ::= \mathbf{tt} \mid \mathbf{ff} \mid a \mid \neg a \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathbf{X}\varphi \\ \mid \mathbf{F}\varphi \mid \mathbf{G}\varphi \mid \varphi\mathbf{U}\varphi \mid \varphi\mathbf{W}\varphi \mid \varphi\mathbf{M}\varphi \mid \varphi\mathbf{R}\varphi \end{aligned}$$

where $a \in Ap$. We denote $\mathit{sf}(\varphi)$ the set of subformulas of φ. A subformula ψ of φ is called *proper* if it is neither a conjunction nor a disjunction, i,e., if the root of its syntax tree is labelled by either a, $\neg a$, or a temporal operator. The satisfaction relation $\models$ between ω-words over the alphabet 2^{Ap} and formulas is inductively defined as follows:

$$\begin{array}{lll} w \models \mathbf{tt} & & \\ w \not\models \mathbf{ff} & & \\ w \models a & \text{iff} & a \in w[0] \\ w \models \neg a & \text{iff} & a \notin w[0] \\ w \models \varphi \wedge \psi & \text{iff} & w \models \varphi \text{ and } w \models \psi \\ w \models \varphi \vee \psi & \text{iff} & w \models \varphi \text{ or } w \models \psi \\ w \models \mathbf{X}\varphi & \text{iff} & w_1 \models \varphi \\ w \models \mathbf{F}\varphi & \text{iff} & \exists k.\, w_k \models \varphi \\ w \models \mathbf{G}\varphi & \text{iff} & \forall k.\, w_k \models \varphi \\ w \models \varphi\mathbf{U}\psi & \text{iff} & \exists k.\, w_k \models \psi \ \text{ and } \ \forall j < k.\, w_j \models \varphi \\ w \models \varphi\mathbf{W}\psi & \text{iff} & w \models \mathbf{G}\varphi \ \text{ or } \ w \models \varphi\mathbf{U}\psi \\ w \models \varphi\mathbf{M}\psi & \text{iff} & \exists k.\, w_k \models \varphi \ \text{ and } \ \forall j \leq k.\, w_j \models \psi \\ w \models \varphi\mathbf{R}\psi & \text{iff} & w \models \mathbf{G}\psi \ \text{ or } \ w \models \varphi\mathbf{M}\psi \end{array}$$

Two formulas are equivalent if they are satisfied by the same words. We also introduce the stronger notion of propositional equivalence:

Definition 2.1 (Propositional Equivalence). Given a formula φ, we assign to it a propositional formula φ_P as follows: replace every maximal proper subformula ψ by a propositional variable x_ψ. Two formulas φ, ψ are *propositionally equivalent*, denoted $\varphi \equiv_P \psi$, iff φ_P and ψ_P are equivalent formulas of propositional logic. The set of all formulas propositionally equivalent to φ is denoted by $[\varphi]_P$.

Example 2.2. Let $\varphi = \mathbf{X}b \vee (\mathbf{G}(a \vee \mathbf{X}b) \wedge \mathbf{X}b)$ with $\psi_1 = \mathbf{X}b$ and $\psi_2 = \mathbf{G}(a \vee \mathbf{X}b)$. We have $\varphi_P = x_{\psi_1} \vee (x_{\psi_2} \wedge x_{\psi_1}) \equiv_P x_{\psi_1}$. Thus $\mathbf{X}b$ is propositionally equivalent to φ and $\mathbf{X}b \in [\varphi]_P$. △

Observe that propositional equivalence implies equivalence, but the converse does not hold.

3 The "after" Function

We recall the definition of the"after function" $af(\varphi, w)$, read "φ after w" [13, 15]. The function assigns to a formula φ and a finite word w another formula such that, intuitively, φ holds for ww' iff $af(\varphi, w)$ holds "after reading w", that is, iff $w' \models af(\varphi, w)$.[1]

Definition 3.1. Let φ be a formula and $\nu \in 2^{Ap}$ a single letter. The formula $af(\varphi, \nu)$ is inductively defined as follows:

$$\begin{aligned} af(a,\nu) &= \begin{cases} \mathbf{tt} & \text{if } a \in \nu \\ \mathbf{ff} & \text{if } a \notin \nu \end{cases} & af(\mathbf{tt},\nu) &= \mathbf{tt} \\ & & af(\mathbf{ff},\nu) &= \mathbf{ff} \\ af(\neg a,\nu) &= \begin{cases} \mathbf{ff} & \text{if } a \in \nu \\ \mathbf{tt} & \text{if } a \notin \nu \end{cases} & af(\varphi \wedge \psi,\nu) &= af(\varphi,\nu) \wedge af(\psi,\nu) \\ & & af(\varphi \vee \psi,\nu) &= af(\varphi,\nu) \vee af(\psi,\nu) \end{aligned}$$

$$\begin{aligned} af(\mathbf{X}\varphi,\nu) &= \varphi \\ af(\mathbf{F}\varphi,\nu) &= af(\varphi,\nu) \vee \mathbf{F}\varphi \\ af(\mathbf{G}\varphi,\nu) &= af(\varphi,\nu) \wedge \mathbf{G}\varphi \\ af(\varphi\mathbf{U}\psi,\nu) &= af(\psi,\nu) \vee (af(\varphi,\nu) \wedge \varphi\mathbf{U}\psi) \\ af(\varphi\mathbf{W}\psi,\nu) &= af(\psi,\nu) \vee (af(\varphi,\nu) \wedge \varphi\mathbf{W}\psi) \\ af(\varphi\mathbf{M}\psi,\nu) &= af(\psi,\nu) \wedge (af(\varphi,\nu) \vee \varphi\mathbf{M}\psi) \\ af(\varphi\mathbf{R}\psi,\nu) &= af(\psi,\nu) \wedge (af(\varphi,\nu) \vee \varphi\mathbf{R}\psi) \end{aligned}$$

Furthermore, we generalize the definition to finite words by setting $af(\varphi, \epsilon) = \varphi$ and $af(\varphi, \nu w) = af(af(\varphi, \nu), w)$ for every $\nu \in 2^{Ap}$ and every finite word w. Finally, we define the set of formulas *reachable from* φ as $Reach(\varphi) = \{[\psi]_P \mid \exists w.\ \psi = af(\varphi, w)\}$.

Example 3.2. Let $\varphi = a \vee (b\ \mathbf{U}\ c)$. We then have $af(\varphi, \{a\}) \equiv_P \mathbf{tt}$, $af(\varphi, \{b\}) \equiv_P (b\ \mathbf{U}\ c)$, $af(\varphi, \{c\}) \equiv_P \mathbf{tt}$, and $af(\varphi, \emptyset) \equiv_P \mathbf{ff}$. △

The following lemma states the main properties of af, which are easily proved by induction on the structure of φ. For convenience we include the short proof in the appendix of [16].

Lemma 3.3. *[15]*

(1) *For every formula φ, finite word $w \in (2^{Ap})^*$, and infinite word $w' \in (2^{Ap})^\omega$: $ww' \models \varphi$ iff $w' \models af(\varphi, w)$*
(2) *For every formula φ and finite word $w \in (2^{Ap})^*$: $af(\varphi, w)$ is a positive boolean combination of proper subformulas of φ.*
(3) *For every formula φ: If φ has n proper subformulas, then $Reach(\varphi)$ has at most size 2^{2^n}.*

It is easy to show by induction that $\varphi \equiv_P \psi$ implies $af(\varphi, w) \equiv_P af(\psi, w)$ for every finite word w. We extend af to equivalence classes by defining $af([\varphi]_P, w) := [af(\varphi, w)]_P$. Sometimes we abuse language and identify a formula and its equivalence class. For example, we write "the states of the automaton are pairs of formulas" instead of "pairs of equivalence classes of formulas".

4 Constructing DRAs for Fragments of LTL

We show that the function af can be used to construct deterministic Büchi and coBüchi automata for some fragments of LTL. The constructions are very simple. Later, in Sections 6, 7, and 8 we use these constructions as building blocks for the translation of general LTL formulas. The fragments are:

- The μ-fragment μLTL and the ν-fragment νLTL.
  μLTL is the fragment of LTL restricted to temporal operators $\mathbf{F}, \mathbf{U}, \mathbf{M}$, on top of Boolean connectives $(\wedge, \vee)$, literals $(a, \neg a)$, and the next operator $(\mathbf{X})$. νLTL is defined analogously, but with the operators $\mathbf{G}, \mathbf{W}, \mathbf{R}$. In the literature μLTL is also called syntactic co-safety and νLTL syntactic safety.
- The fragments $\mathbf{GF}(\mu LTL)$ and $\mathbf{FG}(\nu LTL)$.
  These fragments contain the formulas of the form $\mathbf{GF}\varphi$, where $\varphi \in \mu LTL$, and $\mathbf{FG}\varphi$, where $\varphi \in \nu LTL$.

The reason for the names μLTL and νLTL is that $\mathbf{F}, \mathbf{U}, \mathbf{M}$ are least-fixed-point operators, in the sense that their semantics is naturally formulated by least fixed points, e.g. in the μ-calculus, while the semantics of $\mathbf{G}, \mathbf{W}, \mathbf{R}$ is naturally formulated by greatest fixed points.

The following lemma characterizes the words w satisfying a formula φ of these fragments in terms of the formulas $af(\varphi, w)$.

Lemma 4.1. *[15] Let $\varphi \in \mu LTL$ and let w be a word. We have:*

- $w \models \varphi$ *iff* $\exists i.\ af(\varphi, w_{0i}) \equiv_P \mathbf{tt}$.
- $w \models \mathbf{GF}\varphi$ *iff* $\forall i.\ \exists j.\ af(\mathbf{F}\varphi, w_{ij}) \equiv_P \mathbf{tt}$.

Let $\varphi \in \nu LTL$ and let w be a word. We have:

- $w \models \varphi$ *iff* $\forall i.\ af(\varphi, w_{0i}) \not\equiv_P \mathbf{ff}$.
- $w \models \mathbf{FG}\varphi$ *iff* $\exists i.\forall j.\ af(\mathbf{G}\varphi, w_{ij}) \not\equiv_P \mathbf{ff}$

The following proposition constructs DBAs or DCAs for the fragments. The proof is an immediate consequence of the lemma.

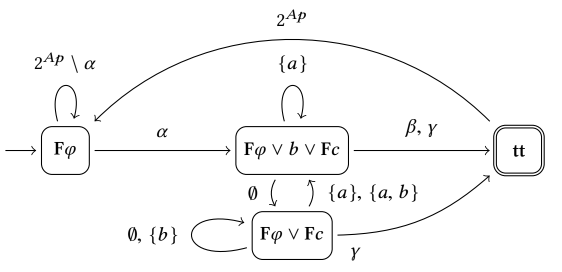
Proposition 4.2. *Let $\varphi \in \mu LTL$.*

- *The following DBA over the alphabet 2^{Ap} recognizes $L(\varphi)$:*
$$\mathcal{A}^\varphi_\mu = (Reach(\varphi), af, \varphi, inf(\mathbf{tt}))$$
- *The following DBA over the alphabet 2^{Ap} recognizes $L(\mathbf{GF}\varphi)$:*
$$\mathcal{A}^\varphi_{\mathbf{GF}\mu} = (Reach(\mathbf{F}\varphi), af_{\mathbf{F}\varphi}, \mathbf{F}\varphi, inf(\mathbf{tt}))$$
$$af_{\mathbf{F}\varphi}(\psi, \nu) = \begin{cases} \mathbf{F}\varphi & \textit{if } \psi \equiv_P \mathbf{tt} \\ af(\psi, \nu) & \textit{otherwise.} \end{cases}$$

Let $\varphi \in \nu LTL$.

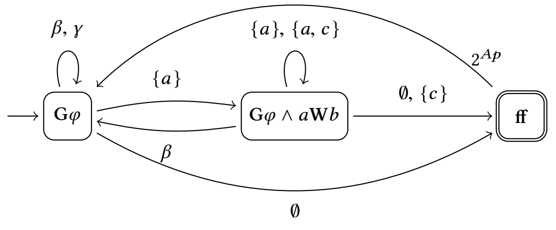
- *The following DCA over the alphabet 2^{Ap} recognizes $L(\varphi)$:*
$$\mathcal{A}^\varphi_\nu = (Reach(\varphi), af, \varphi, fin(\mathbf{ff}))$$
- *The following DCA over the alphabet 2^{Ap} recognizes $L(\mathbf{FG}\varphi)$:*
$$\mathcal{A}^\varphi_{\mathbf{FG}\nu} = (Reach(\mathbf{G}\varphi), af_{\mathbf{G}\varphi}, \mathbf{G}\varphi, fin(\mathbf{ff}))$$
$$af_{\mathbf{G}\varphi}(\psi, \nu) = \begin{cases} \mathbf{G}\varphi & \textit{if } \psi \equiv_P \mathbf{ff} \\ af(\psi, \nu) & \textit{otherwise.} \end{cases}$$

Example 4.3. Let $\varphi = a \wedge \mathbf{X}(b \vee \mathbf{F}c) \in \mu LTL$. The DBA $\mathcal{A}^\varphi_{\mathbf{GF}\mu}$ recognizing $L(\mathbf{GF}\varphi)$ is depicted below. We use the abbreviations $\alpha := \{\nu \in 2^{Ap} \mid a \in \nu\}$, $\beta := \{\nu \in 2^{Ap} \mid b \in \nu\}$, and $\gamma := \{\nu \in 2^{Ap} \mid c \in \nu\}$. △



[1]There is a conceptual correspondences to the derivatives of [7] and af directly connects to the classical "LTL expansion laws" [5]. Furthermore, the yet to be introduced $af^\vee$ relates to [1] in a similar way.

Example 4.4. Let $\varphi = a\mathbf{W}b \vee c \in \nu LTL$. The DCA $\mathcal{A}^{\varphi}_{\mathbf{FG}\nu}$ recognizing $L(\mathbf{FG}\varphi)$ is depicted below. We use the abbreviations of Example 4.3 again.



Now consider the formula $\varphi = \mathbf{FG}(a\mathbf{U}b \vee c)$. It does not belong to any of the fragments due to the deeper alternation of the least- and greatest-fixed-point operators: $\mathbf{F} - \mathbf{G} - \mathbf{U}$. If we construct $\mathcal{A}^{\varphi}_{\mathbf{FG}\nu}$ we obtain a DCA isomorphic to the one above, because $af(\psi_1\mathbf{U}\psi_2, \nu)$ and $af(\psi_1\mathbf{W}\psi_2, \nu)$ are defined in the same way. However, the DCA does not recognize $L(\varphi)$: For example, on the word $\{a\}^{\omega}$, it loops on the middle state and accepts, even though $\{a\}^{\omega} \not\models \varphi$. The reason is that $\mathcal{A}^{\varphi}_{\mathbf{FG}\nu}$ checks that the greatest fixed point holds, and cannot enforce satisfaction of the least-fixed-point formula $a\mathbf{U}b$.

If only we were given a promise that $a\mathbf{U}b$ holds infinitely often, then we could conclude that such a run is accepting. We can actually get such promises: for NBA and LDBA via the non-determinism of the automaton, and for DRA via the "non-determinism" of the acceptance condition. In the next section, we investigate how to utilize such promises (Section 5.3) and how to check whether the promises are fulfilled or not (Section 5.4). △

5 The Master Theorem

We present and prove the Master Theorem: A characterization of the words satisfying a given formula from which we can easily extract deterministic, limit-deterministic, and nondeterministic automata of asymptotically optimal size.

We first provide some intuition with the help of an example. Consider the formula $\varphi = \mathbf{FG}((a\mathbf{R}b) \vee (c\mathbf{U}d))$, which does not belong to any of the fragments in the last section, and a word w. Assume we are promised that along w the μ-subformula $c\mathbf{U}d$ holds infinitely often (this is the case e.g. for $w = (\emptyset\{d\})^{\omega}$). In particular, we then know that d holds infinitely often, and so we can "reduce" $w \models^? \varphi$ to $w \models^? \mathbf{FG}((a\mathbf{R}b) \vee (c\mathbf{W}d))$, which belongs to the fragment $\mathbf{FG}(\nu LTL)$.

Assume now we are promised that $c\mathbf{U}d$ only holds finitely often (for example, because $w = \{d\}^4\{c\}^{\omega}$). Even more, we are promised that along the suffix w_5 the formula $c\mathbf{U}d$ never holds any more. How can we use this advice? First, $w \models^? \varphi$ reduces to $w_5 \models^? af(\varphi, w_{05})$ by the fundamental property of af, Lemma 3.3(1). Further, a little computation shows that $af(\varphi, w_{05}) \equiv_P \varphi$, and so that $w \models^? \varphi$ reduces to $w_5 \models^? \varphi$. Finally, using that $c\mathbf{U}d$ never holds again, we reduce $w \models^? \varphi$ to $w_5 \models^? \mathbf{FG}(a\mathbf{R}b \vee \mathbf{ff}) \equiv_P \mathbf{FG}(a\mathbf{R}b)$ which belongs to the fragment $\mathbf{FG}(\nu LTL)$.

This example suggests a general strategy for solving $w \models^? \varphi$:

- Guess the set of least-fixed-point subformulas of φ that hold infinitely often, denoted by $\mathcal{GF}_w$, and the set of greatest-fixed-point subformulas that hold almost always, denoted by $\mathcal{FG}_w$.
- Guess a *stabilization point* after which the least-fixed-point subformulas outside $\mathcal{GF}_w$ do not hold any more, and the greatest-fixed-point subformulas of $\mathcal{FG}_w$ hold forever.
- Use these guesses to reduce $w \models^? \varphi$ to problems $w \models^? \psi$ for formulas ψ that belong to the fragments introduced in the last section.
- Check that the guesses are correct.

In the rest of the section we develop this strategy. In Section 5.1 we introduce the terminology needed to formalize stabilization. Section 5.2 shows how to use a guess X for $\mathcal{GF}$ or a guess Y for $\mathcal{FG}$ to reduce $w \models^? \varphi$ to a simpler problem $w \models^? \varphi[X]_{\nu}$ or $w \models^? \varphi[Y]_{\mu}$, where $\varphi[X]_{\nu}$ and $\varphi[Y]_{\mu}$ are read as "φ with $\mathbf{GF}$-advice X" and "φ with $\mathbf{FG}$-advice Y", respectively. Section 5.3 shows how to use the advice to decide $w \models^? \varphi$. Section 5.4 shows how to check that the advice is correct. The Master Theorem is stated and proved in Section 5.5.

5.1 μ- and ν-stability.

Fix a formula φ. The set of subformulas of φ of the form $\mathbf{F}\psi$, $\psi_1\mathbf{U}\psi_2$, and $\psi_1\mathbf{M}\psi_2$ is denoted by $\mu(\varphi)$. So, loosely speaking, $\mu(\varphi)$ contains the set of subformulas of φ with a least-fixed-point operator at the top of their syntax tree. Given a word w, we are interested in which of these formulas hold infinitely often, and which ones hold at least once, i.e., we are interested in the sets

$$\mathcal{GF}_w = \{\psi \mid \psi \in \mu(\varphi) \wedge w \models \mathbf{GF}\psi\}$$
$$\mathcal{F}_w = \{\psi \mid \psi \in \mu(\varphi) \wedge w \models \mathbf{F}\psi\}$$

Observe that $\mathcal{GF}_w \subseteq \mathcal{F}_w$. We say that w is μ-stable with respect to φ if $\mathcal{GF}_w = \mathcal{F}_w$.

Example 5.1. For $\varphi = \mathbf{G}a \vee b\mathbf{U}c$ we have $\mu(\varphi) = \{b\mathbf{U}c\}$. Let $w = \{a\}^{\omega}$ and $w' = \{b\}\{c\}\{a\}^{\omega}$. We have $\mathcal{F}_w = \emptyset = \mathcal{GF}_w$ and $\mathcal{GF}_{w'} = \emptyset \subset \{b\mathbf{U}c\} = \mathcal{F}_{w'}$. So w is μ-stable with respect to φ, but w' is not. △

Dually, the set of subformulas of φ of the form $\mathbf{G}\psi$, $\psi_1\mathbf{W}\psi_2$, and $\psi_1\mathbf{R}\psi_2$ is denoted by $\nu(\varphi)$. This time we are interested in whether these formulas hold everywhere or almost everywhere, i.e., in the sets

$$\mathcal{FG}_w = \{\psi \mid \psi \in \nu(\varphi) \wedge w \models \mathbf{FG}\psi\}$$
$$\mathcal{G}_w = \{\psi \mid \psi \in \nu(\varphi) \wedge w \models \mathbf{G}\psi\}$$

(Observe that the question whether a ν-formula like, say, $\mathbf{G}a$, holds once or infinitely often makes no sense, because it holds once iff it holds infnitely often.) We have $\mathcal{FG}_w \supseteq \mathcal{G}_w$, and we say that w is ν-stable with respect to φ if $\mathcal{FG}_w = \mathcal{G}_w$.

Example 5.2. Let φ, w and w' as in Example 5.1. We have $\nu(\varphi) = \{\mathbf{G}a\}$. The word w is ν-stable, but w' is not, because $\mathcal{FG}_{w'} = \{\mathbf{G}a\} \supset \emptyset = \mathcal{G}_{w'}$. △

So not every word is μ-stable or ν-stable. However, as shown by the following lemma, all but finitely many suffixes of a word are μ- and ν-stable.

Lemma 5.3. *For every word w there exist indices $i, j \geq 0$ such that for every $k \geq 0$ the suffix w_{i+k} is μ-stable and the suffix w_{j+k} is ν-stable.*

Proof. We only prove the μ-stability part; the proof of the other part is similar. Since $\mathcal{GF}_{w_i} \subseteq \mathcal{F}_{w_i}$ for every $i \geq 0$, it suffices to exhibit an index i such that $\mathcal{GF}_{w_{i+k}} \supseteq \mathcal{F}_{w_{i+k}}$ for every $k \geq 0$. If $\mathcal{GF}_w \supseteq \mathcal{F}_w$ then we can choose $i := 0$. So assume $\mathcal{F}_w \setminus \mathcal{GF}_w \neq \emptyset$. By definition, every $\psi \in \mathcal{F}_w \setminus \mathcal{GF}_w$ holds only finitely often along w. So for every $\psi \in \mathcal{F}_w \setminus \mathcal{GF}_w$ there exists an index i_{ψ} such that

$w_{i_\psi+k} \not\models \psi$ for every $k \geq 0$. Let $i := \max\{i_\psi \mid \psi \in \mathcal{F}_w\}$, which exists because $\mathcal{F}_w$ is a finite set. It follows $\mathcal{GF}_{w_{i+k}} \supseteq \mathcal{F}_{w_{i+k}}$ for every $k \geq 0$, and so every w_{i+k} is μ-stable. ◻

Example 5.4. Let again $\varphi = \mathbf{G}a \vee b\mathbf{U}c$. The word $w' = \{b\}\{c\}\{a\}^\omega$ is neither μ-stable nor ν-stable, but all suffixes $w'_{(2+k)}$ of w' are both μ-stable and ν-stable. △

5.2 The formulas $\varphi[X]_\nu$ and $\varphi[Y]_\mu$.

We first introduce $\varphi[X]_\nu$. Assume we have to determine if a word w satisfies φ, and we are told that w is μ-stable. Further, we are given the set $X \subseteq \mu(\varphi)$ such that $\mathcal{GF}_w = X = \mathcal{F}_w$. We use this oracle information to reduce the problem $w \models^? \varphi$ to a "simpler" problem $w \models^? \varphi[X]_\nu$, where "simpler" means that $\varphi[X]_\nu$ is a formula of νLTL, for which we already know how to construct automata. In other words, we define a formula $\varphi[X]_\nu \in \nu LTL$ such that $\mathcal{GF}_w = X = \mathcal{F}_w$ implies $w \models \varphi$ iff $w \models \varphi[X]_\nu$. (Observe that $X \subseteq \mu(\varphi)$ but $\varphi[X]_\nu \in \nu LTL$, and so the latter, not the former, is the reason for the ν-subscript in the notation $\varphi[X]_\nu$.)

The definition of $\varphi[X]_\nu$ is purely syntactic, and the intuition behind it is very simple. All the main ideas are illustrated by the following examples, where we assume $\mathcal{GF}_w = X = \mathcal{F}_w$:

- $\varphi = \mathbf{F}a \wedge \mathbf{G}b$ and $X = \{\mathbf{F}a\}$. Then $\mathbf{F}a \in \mathcal{GF}_w$, which implies in particular $w \models \mathbf{F}a$. So we can reduce $w \models^? \mathbf{F}a \wedge \mathbf{G}b$ to $w \models^? \mathbf{G}b$, and so $\varphi[X]_\nu := \mathbf{G}b$.
- $\varphi = \mathbf{F}a \wedge \mathbf{G}b$ and $X = \emptyset$. Then $\mathbf{F}a \notin \mathcal{F}_w$, and so $w \not\models \mathbf{F}a$. So we can reduce $w \models^? \mathbf{F}a \wedge \mathbf{G}b$ to the trivial problem $w \models^? \mathbf{ff}$, and so $\varphi[X]_\nu := \mathbf{ff}$.
- $\varphi = \mathbf{G}(b\mathbf{U}c)$ and $X = \{b\mathbf{U}c\}$. Then $b\mathbf{U}c \in \mathcal{GF}_w$, and so $w \models \mathbf{GF}(b\mathbf{U}c)$. This does not imply $w \models b\mathbf{U}c$, but implies that c will hold in the future. So we can reduce $w \models^? \mathbf{G}(b\mathbf{U}c)$ to $w \models^? \mathbf{G}(b\mathbf{W}c)$, a formula of νLTL, and so $\varphi[X]_\nu := \mathbf{G}(b\mathbf{W}c)$.

Definition 5.5. Let φ be a formula and let $X \subseteq \mu(\varphi)$. The formula $\varphi[X]_\nu$ is inductively defined as follows:

- If $\varphi = \mathbf{tt}, \mathbf{ff}, a, \neg a$, then $\varphi[X]_\nu = \varphi$.
- If $\varphi = op(\psi)$ for $op \in \{\mathbf{X}, \mathbf{G}\}$ then $\varphi[X]_\nu = op(\psi[X]_\nu)$.
- If $\varphi = op(\psi_1, \psi_2)$ for $op \in \{\wedge, \vee, \mathbf{W}, \mathbf{R}\}$ then $\varphi[X]_\nu = op(\psi_1[X]_\nu, \psi_2[X]_\nu)$.
- If $\varphi = \mathbf{F}\psi$ then $\varphi[X]_\nu = \begin{cases} \mathbf{tt} & \text{if } \varphi \in X \\ \mathbf{ff} & \text{otherwise.} \end{cases}$
- If $\varphi = \psi_1\mathbf{U}\psi_2$ then $\varphi[X]_\nu = \begin{cases} (\psi_1[X]_\nu)\mathbf{W}(\psi_2[X]_\nu) & \text{if } \varphi \in X \\ \mathbf{ff} & \text{otherwise.} \end{cases}$
- If $\varphi = \psi_1\mathbf{M}\psi_2$ then $\varphi[X]_\nu = \begin{cases} (\psi_1[X]_\nu)\mathbf{R}(\psi_2[X]_\nu) & \text{if } \varphi \in X \\ \mathbf{ff} & \text{otherwise.} \end{cases}$

We now introduce, in a dual way, a formula $\varphi[Y]_\mu \in \mu LTL$ such that $\mathcal{FG}_w = Y = \mathcal{G}_w$ implies $w \models \varphi$ iff $w \models \varphi[Y]_\mu$.

Definition 5.6. Let φ be a formula and let $Y \subseteq \nu(\varphi)$. The formula $\varphi[Y]_\mu$ is inductively defined as follows:

- If $\varphi = \mathbf{tt}, \mathbf{ff}, a, \neg a$, then $\varphi[Y]_\mu = \varphi$.
- If $\varphi = op(\psi)$ for $op \in \{\mathbf{X}, \mathbf{F}\}$ then $\varphi[Y]_\mu = op(\psi[Y]_\mu)$.
- If $\varphi = op(\psi_1, \psi_2)$ for $op \in \{\wedge, \vee, \mathbf{U}, \mathbf{M}\}$ then $\varphi[Y]_\mu = op(\psi_1[Y]_\mu, \psi_2[Y]_\mu)$.
- If $\varphi = \mathbf{G}\psi$ then $\varphi[Y]_\mu = \begin{cases} \mathbf{tt} & \text{if } \varphi \in Y \\ \mathbf{ff} & \text{otherwise.} \end{cases}$
- If $\varphi = \psi_1\mathbf{W}\psi_2$ then $\varphi[Y]_\mu = \begin{cases} \mathbf{tt} & \text{if } \varphi \in Y \\ (\psi_1[Y]_\mu)\mathbf{U}(\psi_2[Y]_\mu) & \text{otherwise.} \end{cases}$
- If $\varphi = \psi_1\mathbf{R}\psi_2$ then $\varphi[Y]_\mu = \begin{cases} \mathbf{tt} & \text{if } \varphi \in Y \\ (\psi_1[Y]_\mu)\mathbf{M}(\psi_2[Y]_\mu) & \text{otherwise.} \end{cases}$

Example 5.7. Let $\varphi = ((a\mathbf{W}b) \wedge \mathbf{F}c) \vee a\mathbf{U}d$. We have:

$$\begin{array}{llll}
\varphi[\{\mathbf{F}c\}]_\nu & = ((a\mathbf{W}b) \wedge \mathbf{tt}) \vee \mathbf{ff} & \equiv_P & a\mathbf{W}b \\
\varphi[\{a\mathbf{U}d\}]_\nu & = ((a\mathbf{W}b) \wedge \mathbf{ff}) \vee a\mathbf{W}d & \equiv_P & a\mathbf{W}d \\
\varphi[\emptyset]_\nu & = ((a\mathbf{W}b) \wedge \mathbf{ff}) \vee \mathbf{ff} & \equiv_P & \mathbf{ff} \\
\varphi[\{a\mathbf{W}b\}]_\mu & = (\mathbf{tt} \wedge \mathbf{F}c) \vee a\mathbf{U}d & \equiv_P & \mathbf{F}c \vee a\mathbf{U}d \\
\varphi[\emptyset]_\mu & = (a\mathbf{U}b \wedge \mathbf{F}c) \vee a\mathbf{U}d & &
\end{array}$$

△

5.3 Utilizing $\varphi[X]_\nu$ and $\varphi[Y]_\mu$.

The following lemma states the fundamental properties of $\varphi[X]_\nu$ and $\varphi[Y]_\mu$. As announced above, for a μ-stable word w we can reduce the problem $w \models^? \varphi$ to $w \models^? \varphi[X]_\nu$, and for a ν-stable word to $w \models^? \varphi[Y]_\mu$. However, there is more: If we only know $X \subseteq \mathcal{GF}_w$, then we can still infer $w \models \varphi$ from $w \models \varphi[X]_\nu$, only the implication in the other direction fails.

Lemma 5.8. *Let φ be a formula and let w be a word.*
For every $X \subseteq \mu(\varphi)$:

(a1) *If $\mathcal{F}_w \subseteq X$ and $w \models \varphi$, then $w \models \varphi[X]_\nu$.*
(a2) *If $X \subseteq \mathcal{GF}_w$ and $w \models \varphi[X]_\nu$, then $w \models \varphi$.*

In particular:

(a3) *If $\mathcal{F}_w = X = \mathcal{GF}_w$ then $w \models \varphi$ iff $w \models \varphi[X]_\nu$.*

For every $Y \subseteq \nu(\varphi)$:

(b1) *If $\mathcal{FG}_w \subseteq Y$ and $w \models \varphi$, then $w \models \varphi[Y]_\mu$.*
(b2) *If $Y \subseteq \mathcal{G}_w$ and $w \models \varphi[Y]_\mu$, then $w \models \varphi$.*

In particular:

(b3) *If $\mathcal{FG}_w = Y = \mathcal{G}_w$ then $w \models \varphi$ iff $w \models \varphi[Y]_\mu$.*

Proof. All parts are proved by a straightforward structural induction on φ. We consider only (a1), and only two representative cases of the induction. Representative cases for (a2), (b1), and (b2) can be found in the appendix of [16].

(a1) Assume $\mathcal{F}_w \subseteq X$. Then $\mathcal{F}_{w_i} \subseteq X$ for all $i \geq 0$. We prove the following stronger statement via structural induction on φ:

$$\forall i.\ (\ (w_i \models \varphi) \rightarrow (w_i \models \varphi[X]_\nu)\ )$$

We consider one representative of the "interesting" cases, and one of the "straightforward" cases.

Case $\varphi = \psi_1\mathbf{U}\psi_2$: Let $i \geq 0$ arbitrary and assume $w_i \models \psi_1\mathbf{U}\psi_2$. Then $\psi_1\mathbf{U}\psi_2 \in \mathcal{F}_{w_i}$ and so $\varphi \in X$. We prove $w_i \models (\psi_1\mathbf{U}\psi_2)[X]_\nu$:

$$\begin{array}{lll}
& w_i \models \psi_1\mathbf{U}\psi_2 & \\
\Longrightarrow & w_i \models \psi_1\mathbf{W}\psi_2 & \\
\Longrightarrow & \forall j.\ w_{i+j} \models \psi_1 \vee \exists k \leq j.\ w_{i+k} \models \psi_2 & \\
\Longrightarrow & \forall j.\ w_{i+j} \models \psi_1[X]_\nu \vee \exists k \leq j.\ w_{i+k} \models \psi_2[X]_\nu & \text{(I.H.)} \\
\Longrightarrow & w_i \models (\psi_1[X]_\nu)\mathbf{W}(\psi_2[X]_\nu) & \\
\Longrightarrow & w_i \models (\psi_1\mathbf{U}\psi_2)[X]_\nu & (\varphi \in X, \text{Def. 5.5})
\end{array}$$

Case $\varphi = \psi_1 \vee \psi_2$: Let $i \geq 0$ arbitrary and assume $w_i \models \psi_1 \vee \psi_2$:

$$\begin{array}{lll}
& w_i \models \psi_1 \vee \psi_2 & \\
\Longrightarrow & (w_i \models \psi_1) \vee (w_i \models \psi_2) & \\
\Longrightarrow & (w_i \models \psi_1[X]_\nu) \vee (w_i \models \psi_2[X]_\nu) & \text{(I.H.)} \\
\Longrightarrow & w_i \models (\psi_1 \vee \psi_2)[X]_\nu & \text{(Def. 5.5)}
\end{array}$$

◻

Lemma 5.8 suggests to decide $w \models^? \varphi$ by "trying out" all possible sets X. Part (a2) shows that the strategy of checking for every set X if both $X \subseteq \mathcal{GF}_w$ and $w \models \varphi[X]_\nu$ hold is sound.

Example 5.9. Consider $\varphi = \mathbf{GF}a \vee \mathbf{GF}(b \wedge \mathbf{G}c)$. Since $\mu(\varphi) = \{\mathbf{F}a, \mathbf{F}(b \wedge \mathbf{G}c)\}$, there are four possible X's to be tried out: $\emptyset$, $\{\mathbf{F}a\}$, $\{\mathbf{F}(b \wedge \mathbf{G}c)\}$, and $\{\mathbf{F}a, \mathbf{F}(b \wedge \mathbf{G}c)\}$. For $X = \emptyset$ we get $\varphi[X]_\nu = \mathbf{ff}$, indicating that if neither a nor $b \wedge \mathbf{G}c$ hold infinitely often, then φ cannot hold. For the other three possibilities (a holds infinitely often, $b \wedge \mathbf{G}c$ holds infinitely often, or both) there are words satisfying φ, like a^ω, $\{b, c\}^\omega$, and $\{a, b, c\}^\omega$. △

However there are still two questions open. First, is this strategy complete? Part (a3) shows that it is complete for μ-stable words: Indeed, in this case there is a set X such that $\mathcal{GF}_w = X = \mathcal{F}_w$, and for this particular set $w \models \varphi[X]_\nu$ holds. For words that are not μ-stable, we will use the existence of μ-stable suffixes: Instead of checking $w \models \varphi[X]_\nu$, we will check the existence of a suffix w_i such that $w_i \models af(\varphi, w_{0i})[X]_\nu$. This will happen in Section 5.5. The second open question is simply how to check $X \subseteq \mathcal{GF}_w$. We deal with it in Section 5.4.

5.4 Checking $X \subseteq \mathcal{GF}_w$ and $Y \subseteq \mathcal{FG}_w$.

Consider again the formula $\varphi = \mathbf{GF}a \vee \mathbf{GF}(b \wedge \mathbf{G}c)$ of Example 5.9. If $X = \{\mathbf{F}a\}$, then checking whether X is a correct advice (i.e., whether $X \subseteq \mathcal{GF}_w$ holds) is easy, because $\mathbf{GFF}a \in \mathbf{GF}(\mu LTL)$, see Proposition 4.2. In contrast, for $X = \{\mathbf{F}(b \wedge \mathbf{G}c)\}$ this is not so. In this case it would come handy if we had an advice $Y = \{\mathbf{G}c\}$ promising that $\mathbf{G}c$ holds almost always, as is the case for e.g. $\emptyset^5(\{b, c\}\{c\})^\omega$. Indeed, we could easily check correctness of this advice, because $\mathbf{FGG}c \in \mathbf{FG}(\nu LTL)$, and with its help checking $\mathbf{GF}(b \wedge \mathbf{G}c)$ reduces to checking $\mathbf{GF}(b \wedge \mathbf{tt}) = \mathbf{GF}b$, which is also easy.

One of the main ingredients of our approach is that in order to verify a promise $X \subseteq \mathcal{GF}_w$ we can rely on a promise $Y \subseteq \mathcal{FG}_w$ about subformulas of X, and vice versa. There is no circularity in this rely/guarantee reasoning because the subformula order is well founded, and we eventually reach formulas ψ such that $\psi[X]_\nu = \psi$ or $\psi[Y]_\mu = \psi$. This argument is formalized in the next lemma. The first part of the lemma states that mutually assuming correctness of the other promise is correct. The second part states that, loosely speaking, this rely/guarantee method is complete.

Lemma 5.10. *Let φ be a formula and let w be a word.*

(1.) *For every $X \subseteq \mu(\varphi)$ and $Y \subseteq \nu(\varphi)$, if*

$$\forall \psi \in X.\ w \models \mathbf{GF}(\psi[Y]_\mu)$$
$$\forall \psi \in Y.\ w \models \mathbf{FG}(\psi[X]_\nu)$$

then $X \subseteq \mathcal{GF}_w$ and $Y \subseteq \mathcal{FG}_w$.

(2.) *If $X = \mathcal{GF}_w$ and $Y = \mathcal{FG}_w$ then*

$$\forall \psi \in X.\ w \models \mathbf{GF}(\psi[Y]_\mu)$$
$$\forall \psi \in Y.\ w \models \mathbf{FG}(\psi[X]_\nu)$$

Proof. (1.) Let $X \subseteq \mu(\varphi)$ and $Y \subseteq \nu(\varphi)$. Observe that $X \cap Y = \emptyset$. Let $n := |X \cup Y|$. Let $\psi_1, \ldots, \psi_n$ be an enumeration of $X \cup Y$ compatible with the subformula order, i.e., if ψ_i is a subformula of ψ_j, then $i \leq j$. Finally, let $(X_0, Y_0), (X_1, Y_1), \ldots, (X_n, Y_n)$ be the unique sequence of pairs satisfying:

- $(X_0, Y_0) = (\emptyset, \emptyset)$ and $(X_n, Y_n) = (X, Y)$.
- For every $0 < i \leq n$, if $\psi_i \in X$ then $X_i \setminus X_{i-1} = \{\psi_i\}$ and $Y_i = Y_{i-1}$, and if $\psi_i \in Y$, then $X_i = X_{i-1}$ and $Y_i \setminus Y_{i-1} = \{\psi_i\}$.

We prove $X_i \subseteq \mathcal{GF}_w$ and $Y_i \subseteq \mathcal{FG}_w$ for every $0 \leq i \leq n$ by induction on i. For $i = 0$ the result follows immediately from $X_0 = \emptyset = Y_0$. For $i > 0$ we consider two cases:

Case 1: $\psi_i \in Y$, i.e., $X_i = X_{i-1}$ and $Y_i \setminus Y_{i-1} = \{\psi_i\}$.

By induction hypothesis and $X_i = X_{i-1}$ we have $X_i \subseteq \mathcal{GF}_w$ and $Y_{i-1} \subseteq \mathcal{FG}_w$. We prove $\psi_i \in \mathcal{FG}_w$, i.e., $w \models \mathbf{FG}\psi_i$, in three steps.

Claim 1: $\psi_i[X]_\nu = \psi_i[X_i]_\nu$.

By the definition of the $\cdot[\cdot]_\nu$ mapping, $\psi_i[X]_\nu$ is completely determined by the μ-subformulas of ψ_i that belong to X. By the definition of the sequence $(X_0, Y_0), \ldots, (X_n, Y_n)$, a μ-subformula of ψ_i belongs to X iff it belongs to X_i, and we are done.

Claim 2: $X_i \subseteq \mathcal{GF}_{w_k}$ for every $k \geq 0$.

Follows immediately from $X_i \subseteq \mathcal{GF}_w$.

Proof of $w \models \mathbf{FG}\psi_i$**.** By the assumption of the lemma we have $w \models \mathbf{FG}(\psi_i[X]_\nu)$, and so, by Claim 1, $w \models \mathbf{FG}(\psi_i[X_i]_\nu)$. So there exists an index j such that $w_{j+k} \models \psi_i[X_i]_\nu$ for every $k \geq 0$. By Claim 2 we further have $X_i \subseteq \mathcal{GF}_{w_{j+k}}$ for every $j, k \geq 0$. So we can apply part (a2) of Lemma 5.8 to X_i, w_{j+k}, and ψ_i, which yields $w_{j+k} \models \psi_i$ for every $k \geq 0$. So $w \models \mathbf{FG}\psi_i$.

Case 2: $\psi_i \in X$, i.e., $X_i \setminus X_{i-1} = \{\psi_i\}$ and $Y_i = Y_{i-1}$.

In this case $X_{i-1} \subseteq \mathcal{GF}_w$ and $Y_i \subseteq \mathcal{FG}_w$. We prove $\psi_i \in \mathcal{GF}_w$, i.e., $w \models \mathbf{GF}\psi_i$ in three steps.

Claim 1: $\psi_i[Y]_\mu = \psi_i[Y_i]_\mu$.

The claim is proved as in Case 1.

Claim 2: There is an $j \geq 0$ such that $Y_i \subseteq \mathcal{G}_{w_k}$ for every $k \geq j$.

Follows immediately from $Y_i \subseteq \mathcal{FG}_w$.

Proof of $w \models \mathbf{GF}\psi_i$**.** By the assumption of the lemma we have $w \models \mathbf{GF}(\psi_i[Y]_\mu)$. Let j be the index of Claim 2. By Claim 1 we have $w \models \mathbf{GF}(\psi_i[Y_i]_\mu)$, and so there exist infinitely many $k \geq j$ such that $w_k \models \psi_i[Y_i]_\mu$. By Claim 2 we further have $Y_i \subseteq \mathcal{G}_{w_k}$. So we can apply part (b2) of Lemma 5.8 to Y_i, w_k, and ψ_i, which yields $w_k \models \psi_i$ for infinitely many $k \geq j$. So $w \models \mathbf{GF}\psi_i$.

(2.) Let $\psi \in \mathcal{GF}_w$. We have $w \models \mathbf{GF}\psi$, and so $w_i \models \psi$ for infinitely many $i \geq 0$. Since $\mathcal{FG}_{w_i} = \mathcal{FG}_w$ for every $i \geq 0$, part (b1) of Lemma 5.8 can be applied to w_i, $\mathcal{FG}_{w_i}$, and ψ. This yields $w_i \models \psi[\mathcal{FG}_w]_\mu$ for infinitely many $i \geq 0$ and thus $w \models \mathbf{GF}(\psi[\mathcal{FG}_w]_\mu)$.

Let $\psi \in \mathcal{FG}_w$. Since $w_i \models \mathbf{FG}\psi$, there is an index j such that $w_{j+k} \models \psi$ for every $k \geq 0$. By Lemma 5.3 the index j can be chosen so that it also satisfies $\mathcal{GF}_w = \mathcal{F}_{w_{j+k}} = \mathcal{GF}_{w_{j+k}}$ for every $k \geq 0$. So part (a1) of Lemma 5.8 can be applied to $\mathcal{F}_{w_{j+k}}$, w_{j+k}, and ψ. This yields $w_{j+k} \models \psi[\mathcal{GF}_w]_\nu$ for every $k \geq 0$ and thus $w \models \mathbf{FG}(\psi[\mathcal{GF}_w]_\nu)$. □

Example 5.11. Let $\varphi = \mathbf{F}(a \wedge \mathbf{G}(b \vee \mathbf{F}c))$, $X = \{\varphi\}$, and $Y = \{\mathbf{G}(b \vee \mathbf{F}c)\}$.

- The condition $\forall \psi \in X.\ w \models \mathbf{GF}(\psi[Y]_\mu)$ becomes
$$w \models \mathbf{GF}\left(\varphi[Y]_\mu\right) = \mathbf{GF}(\mathbf{F}a) \equiv \mathbf{GF}a$$
- The condition $\forall \psi \in Y.\ w \models \mathbf{FG}(\psi[X]_\nu)$ becomes
$$w \models \mathbf{FG}\left(\mathbf{G}(b \vee \mathbf{F}c)[X]_\nu\right) = \mathbf{FG}(\mathbf{G}b) \equiv \mathbf{FG}b$$

By Lemma 5.10 (1) we then have that $w \models \mathbf{GF}a \wedge \mathbf{FG}b$ implies $\varphi \in \mathcal{GF}_w$ and $\mathbf{G}(b \vee \mathbf{F}c)) \in \mathcal{FG}_w$. △

5.5 Putting the pieces together: The Master Theorem.

Putting together Lemma 5.8 and Lemma 5.10, we obtain the main result of the paper, which we will use as "Master Theorem" for the construction of automata.

Theorem 5.12 (Master Theorem). *For every formula φ and for every word w: $w \models \varphi$ iff there exists $X \subseteq \mu(\varphi)$ and $Y \subseteq \nu(\varphi)$ satisfying*

$$\begin{aligned}&(1) \quad \exists i.\ w_i \models \mathit{af}(\varphi, w_{0i})[X]_\nu\\&(2) \quad \forall \psi \in X.\ w \models \mathbf{GF}(\psi[Y]_\mu)\\&(3) \quad \forall \psi \in Y.\ w \models \mathbf{FG}(\psi[X]_\nu)\end{aligned}$$

Observe that $\mathit{af}(\varphi, w_{0i})[X]_\nu$, $\mathbf{GF}(\psi[Y]_\mu)$, and $\mathbf{FG}(\psi[X]_\nu)$ are formulas of νLTL, $\mathbf{GF}(\mu LTL)$, and $\mathbf{FG}(\nu LTL)$, respectively, i.e., they all belong to the fragments of Section 4.

Before proving the theorem, let us interpret it in informal terms. The Master Theorem states that in order to decide $w \models^? \varphi$ we can guess two sets $X \subseteq \mu(\varphi)$ and $Y \subseteq \nu(\varphi)$ and an index i, and then proceed as follows: verify $Y \subseteq \mathcal{FG}_w$ assuming that $X \subseteq \mathcal{GF}_w$ holds (3), verify $X \subseteq \mathcal{GF}_w$ assuming that $Y \subseteq \mathcal{FG}_w$ holds (2), and verify $w_i \models \mathit{af}(\varphi, w_{0i})$ assuming that $X \subseteq \mathcal{GF}_w$ holds (1). The procedure is sound by Lemma 5.8 and Lemma 5.10, and complete because the guess where $X := \mathcal{GF}_w$, $Y := \mathcal{FG}_w$, and i is a stabilization point of w, is guaranteed to succeed.

Example 5.13. Let $\varphi = \mathbf{F}(a \wedge \mathbf{G}(b \vee \mathbf{F}c))$ as in Example 5.11, and let $\varphi' = d\mathbf{U}\varphi$. For $X = \{\varphi, \varphi'\}$, $Y = \{\mathbf{G}(b \vee \mathbf{F}c)\}$, and $i = 0$ the Master Theorem yields that $w \models \varphi'$ is implied by

(1) $w \models (d\mathbf{U}\varphi)[X]_\nu = d\mathbf{W}(\varphi[X]_\nu) = d\mathbf{W}\mathbf{tt} \equiv \mathbf{tt}$,
(2) $w \models \mathbf{GF}(\varphi[Y]_\mu) \wedge \mathbf{GF}(\varphi'[Y]_\mu) = \mathbf{GF}a \wedge \mathbf{GF}(d\mathbf{U}(\mathbf{F}a))$, and
(3) $w \models \mathbf{FG}((\mathbf{G}(b \vee \mathbf{F}c)[X]_\nu) \equiv \mathbf{FG}b$.

For $X = \{\varphi\}$, $Y = \{\mathbf{G}(b \vee \mathbf{F}c)\}$, and $i = 0$, condition (1) is $w \models \mathbf{ff}$, and we do not derive any useful information. △

Proof (of the Master Theorem).
(⇒): Assume $w \models \varphi$, and set $X := \mathcal{GF}_w$ and $Y := \mathcal{FG}_w$. Properties (2) and (3) follow from Lemma 5.10. For property (1), let i be an index such that $\mathcal{F}_{w_i} = \mathcal{GF}_{w_i}$; this index exists by Lemma 5.3. By Lemma 3.3 we have $w_i \models \mathit{af}(\varphi, w_{0i})$, and by Lemma 5.8 (a1) $w_i \models \mathit{af}(\varphi, w_{0i})[X]_\nu$.
(⇐): Assume that properties (1-3) hold for sets X, Y and an index i. By Lemma 5.10 (1.) we have $X \subseteq \mathcal{GF}_w$, and so $X \subseteq \mathcal{GF}_{w_i}$. By Lemma 5.8 (a2) we obtain $w_i \models \mathit{af}(\varphi, w_{0i})[X]_\nu$, and thus $w_i \models \mathit{af}(\varphi, w_{0i})$. Lemma 3.3 yields $w \models \varphi$. □

Let $L^j_{X,Y}$ be the language of all words that satisfy condition (j) of the Master Theorem for the sets X and Y. The Master Theorem can then be reformulated as:

$$L(\varphi) = \bigcup_{\substack{X \subseteq \mu(\varphi)\\ Y \subseteq \nu(\varphi)}} L^1_{X,Y} \cap L^2_{X,Y} \cap L^3_{X,Y}$$

Therefore, given an automata model effectively closed under union and intersection, in order to construct automata for all of LTL it suffices to exhibit automata recognizing $L^1_{X,Y}, L^2_{X,Y}, L^3_{X,Y}$. In the next section we consider the case of DRAs, and then we proceed to NBAs and LDBAs.

6 Constructing DRAs for LTL Formulas

Let φ be a formula of length n. We use the Master Theorem to construct a DRA for $L(\varphi)$ with $2^{2^{O(n)}}$ states and $O(2^n)$ Rabin pairs. Since our purpose is only to show that we can easily obtain automata of asymptotically optimal size, we give priority to a simpler construction over one with the least number of states. We comment in Section 9 on optimizations that reduce the size by using other acceptance conditions.

We first construct DRAs for $L^1_{X,Y}$, $L^2_{X,Y}$, and $L^3_{X,Y}$ with $2^{2^{O(n)}}$ states and one single Rabin pair. More precisely, for each of these languages we construct either a DBA or a DCA. We then construct a DRA for $L(\varphi)$ by means of intersections and unions.

A DCA for $L^1_{X,Y}$. We define a DCA $C_{\varphi,X}$ that accepts a word w iff $w_i \models \mathit{af}(\varphi, w_{0i})[X]_\nu$ for some suffix w_i of w. In the rest of this part of the section we abbreviate $\mathit{af}(\varphi, w_{0i})$ to φ_i. Recall that $\varphi_i[X]_\nu$ is a formula of νLTL, and so for every $i \geq 0$ there is a DCA with a state $\mathbf{ff}$ such that the automaton rejects iff it reaches this state. Intuitively, if the automaton rejects, then it rejects "after finite time". We prove the following lemma:

Lemma 6.1. *Let $\varphi_i := \mathit{af}(\varphi, w_{0i})$. If $w \models \varphi[X]_\nu$ then $w_i \models \varphi_i[X]_\nu$ for all $i > 0$.*

Proof. Assume $w \models \varphi[X]_\nu$. It suffices to prove $w_1 \models \varphi_1[X]_\nu$, since the general case follows immediately by induction. For $i = 1$ we proceed by structural induction on φ, and consider only some representative cases.
Case $\varphi = a$. Since $w \models a[X]_\nu = a$ we have $a \in w[0]$. So $\varphi_1[X]_\nu = \mathbf{tt}[X]_\nu = \mathbf{tt}$, and thus $w_1 \models \varphi_1[X]_\nu$.
Case $\varphi = \psi\mathbf{U}\chi$. Since $w \models \varphi[X]_\nu$ we have $\varphi[X]_\nu \neq \mathbf{ff}$, and so $\varphi \in X$. We have:

$$\begin{aligned}& w \models \varphi[X]_\nu &\\ \Longrightarrow\ & w \models (\psi[X]_\nu)\mathbf{W}(\chi[X]_\nu) & \text{(Def. 5.5)}\\ \Longrightarrow\ & w \models (\psi[X]_\nu \wedge \mathbf{X}((\psi[X]_\nu)\mathbf{W}(\chi[X]_\nu))) \vee \chi[X]_\nu &\\ \Longrightarrow\ & w \models (\psi[X]_\nu \wedge \mathbf{X}((\psi\mathbf{U}\chi)[X]_\nu)) \vee \chi[X]_\nu & (\varphi \in X)\\ \Longrightarrow\ & w_1 \models (\psi_1[X]_\nu \wedge \varphi[X]_\nu) \vee \chi_1[X]_\nu & \text{(I.H.)}\\ \Longrightarrow\ & w_1 \models ((\psi_1 \wedge (\psi\mathbf{U}\chi)) \vee \chi_1)[X]_\nu & \text{(Def. 5.5)}\\ \Longrightarrow\ & w_1 \models \varphi_1[X]_\nu & \text{(Def. 3.1)}\end{aligned}$$

□

Loosely speaking, $C_{\varphi,X}$ starts by checking $w \models^? \varphi[X]_\nu$. For this it maintains the formula $(\varphi[X]_\nu)_i$ in its state. If the formula becomes $\mathbf{ff}$ after, say, j steps, then $w \not\models \varphi[X]_\nu$, and $C_{\varphi,X}$ proceeds to check $w \models^? \varphi_j[X]_\nu$. In order to "switch" to this new problem, $C_{\varphi,X}$ needs to know φ_j, and so it maintains φ_j it in its state. In other words, after j steps $C_{\varphi,X}$ is in state $(\varphi_j, \mathit{af}(\varphi_i[X]_\nu, w_{ij}))$, where $i \leq j$ is the number of steps after which $C_{\varphi,X}$ switched to a new problem for the last time. If the second component of the state becomes $\mathbf{ff}$, then the automaton uses the first component to determine which formula to check next. The accepting condition states that the transitions leading to a state of the form $(\psi, \mathbf{ff})$ must occur finitely often, which implies that eventually one of the checks $w \models^? \varphi_j[X]_\nu$ succeeds.

The formal description of $C_{\varphi,X}$ is as follows:

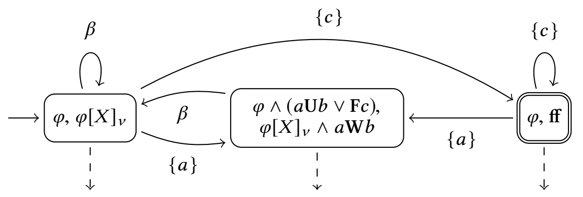
$$C_{\varphi,X} = (\mathit{Reach}(\varphi) \times \mathit{Reach}(\varphi)[X]_\nu, \delta, (\varphi, \varphi[X]_\nu), \mathit{fin}(F))$$

where

- $\mathit{Reach}(\varphi)[X]_\nu = \bigcup_{\psi \in \mathit{Reach}(\varphi)} \mathit{Reach}(\psi[X]_\nu)$
- $\delta((\xi, \zeta), \nu) = \begin{cases}(\mathit{af}(\xi, \nu), \mathit{af}(\xi[X]_\nu, \nu)) & \text{if } \zeta \equiv_P \mathbf{ff}\\ (\mathit{af}(\xi, \nu), \mathit{af}(\zeta, \nu)) & \text{otherwise.}\end{cases}$
- $F = \mathit{Reach}(\varphi) \times \{\mathbf{ff}\}$

Since $\mathit{Reach}(\varphi)$ has at most size 2^{2^n}, the number of states of $C_{\varphi,X}$ is bounded by $\left(2^{2^n}\right)^2 = 2^{O(2^n)}$.

Example 6.2. Let $\varphi = \mathbf{G}(a\mathbf{U}b \vee \mathbf{F}c)$, $X = \{a\mathbf{U}b\}$, and $\varphi[X]_\nu = \mathbf{G}(a\mathbf{W}b)$. Below we show a fragment of $\mathcal{C}_{\varphi,X}$, with α, β, γ as in Example 4.3.



β
$\{c\}$
$\{c\}$
$\varphi, \varphi[X]_\nu$
β
$\varphi \wedge (a\mathbf{U}b \vee \mathbf{F}c)$, $\varphi[X]_\nu \wedge a\mathbf{W}b$
$\varphi, \mathbf{ff}$
$\{a\}$
$\{a\}$

For $w = \{c\}\{c\}(\{a\}\{b\})^\omega$ we have $X = \mathcal{GF}_w$; the word is accepted. For $w' = \{c\}^\omega$ we have $X \neq \mathcal{GF}_{w'}$, and the word is rejected. △

A DBA for $L^2_{X,Y}$**.** We define a DBA recognizing $L\left(\bigwedge_{\psi\in X} \mathbf{GF}(\psi[Y]_\mu)\right)$. Observe that $\mathbf{GF}(\psi[Y]_\mu) \in \mathbf{GF}(\mu LTL)$ for every $\psi \in X$, and that $\psi[Y]_\mu$ has at most n subformulas. By Proposition 4.2, $L(\mathbf{GF}(\psi[Y]_\mu)$ is recognized by a DBA with at most $2^{2^{O(n)}}$ states. Recall that the intersection of the languages of k DBAs with $s_1, \ldots, s_k$ states is recognized by a DBA with $k \cdot \prod_{j=1}^k s_j$ states. Since $|X| \leq n$, the intersection of the DBAs for the formulas $\mathbf{GF}(\psi[Y]_\mu)$ yields a DBA with at most $n \cdot \left(2^{2^{O(n)}}\right)^n = 2^{n2^{O(n)}} = 2^{2^{O(n)}}$ states.

A DCA for $L^3_{X,Y}(\varphi)$**.** The DCA for $L\left(\bigwedge_{\psi\in Y} \mathbf{FG}(\psi[X]_\nu\right)$ is obtained dually to the previous case, applying $\mathbf{FG}(\psi[X]_\nu) \in \mathbf{FG}(\nu LTL)$, and Proposition 4.2.

A DRA for $L(\varphi)$**.** By the Master Theorem we have:

$$L(\varphi) = \bigcup_{\substack{X\subseteq\mu(\varphi)\\ Y\subseteq\nu(\varphi)}} L^1_{X,Y} \cap L^2_{X,Y} \cap L^3_{X,Y}$$

We first construct a DRA $A_{X,Y}$ for the intersection of $L^i_{X,Y}$, where $i = 1, 2, 3$. Let $A^i_{X,Y}$ be the DCA or DBA for $L^i_{X,Y}$. The set of states of $A_{X,Y}$ is the cartesian product of the sets of states of the $A^i_{X,Y}$, the transition function is as usual, and the accepting condition is

$$\mathit{fin}\ ((S_1 \times Q_2 \times Q_3) \cup (Q_1 \times Q_2 \times S_3)) \wedge \mathit{inf}(Q_1 \times S_2 \times Q_3)$$

where Q_i is the set of states of $A^i_{X,Y}$, and $\mathit{fin}(S_1)$, $\mathit{inf}(S_2)$, $\mathit{fin}(S_3)$ are the accepting conditions of $A^1_{X,Y}$, $A^2_{X,Y}$, and $A^3_{X,Y}$.

We construct a DRA A_φ for $L(\varphi)$. Since $X \subseteq \mu(\varphi)$ and $Y \subseteq \nu(\varphi)$, there are at most 2^n pairs of sets X, Y. Let $A_1, \ldots, A_k$ be an enumeration of the DRAs for these pairs, where $k \leq 2^n$, and let Q_i and $\alpha_i = \mathit{fin}(U_i) \wedge \mathit{inf}(V_i)$ be the set of states and accepting condition of A_i, respectively. The set of states of A_φ is $Q_1 \times \cdots \times Q_k$, the transition function is as usual, and the accepting condition is

$$\begin{array}{ll} \bigvee_{i=1}^k & \mathit{fin}(Q_1 \times \cdots \times Q_{i-1} \times U_i \times Q_{i+1} \times \cdots \times Q_k) \quad \wedge \\ & \mathit{inf}(Q_1 \times \cdots \times Q_{i-1} \times V_i \times Q_{i+1} \times \cdots \times Q_k) \end{array}$$

So A_φ has $\left(2^{2^{O(n)}}\right)^{2^n} = 2^{2^{O(n)}\cdot 2^n} = 2^{2^{O(n)}}$ states and at most 2^n Rabin pairs.

7 Constructing NBAs for LTL Formulas

Assume that φ has length n. We use the Master Theorem to construct a NBA for $L(\varphi)$ with $2^{O(n)}$ states.

We first describe how to construct NBAs for the LTL fragments of Section 4. Let us start with some informal intuition. Consider the formula $\varphi = \mathbf{GX}(a \vee b)$. In the DRA for φ we find states for the formulas φ and $\mathit{af}(\varphi, \emptyset)$ and a transition

$$\varphi \xrightarrow{\emptyset} \mathit{af}(\varphi, \emptyset)$$

where $\mathit{af}(\varphi, \emptyset) \equiv_P \varphi \wedge (a \vee b)$. The languages recognized from the states φ and $\mathit{af}(\varphi, \emptyset)$ are precisely $L(\varphi)$ and $L(\mathit{af}(\varphi, \emptyset))$. The basic principle for the construction of the NBAs is to put $\mathit{af}(\varphi, \emptyset)$ in disjunctive normal form (DNF)

$$\varphi \wedge (a \vee b) \equiv_P (\varphi \wedge a) \vee (\varphi \wedge b)$$

and instead of a single transition, have two transitions

$$\varphi \xrightarrow{\emptyset} \varphi \wedge a \quad \text{ and } \quad \varphi \xrightarrow{\emptyset} \varphi \wedge b.$$

In other words, the nondeterminism is used to guess which of the two disjuncts of the DNF is going to hold. Formally, we proceed as follows:

Definition 7.1. We define $\mathit{dnf}(\varphi)$ as the set of clauses obtained by putting the propositional formula φ in DNF, i.e., $\varphi \equiv_P \bigvee_{\psi\in\mathit{dnf}(\varphi)} \psi$. Further let

$$\mathit{Reach}^\vee(\varphi) = \bigcup_{w\in(2^{Ap})^*} \mathit{af}^\vee(\psi, w)$$

with $\mathit{af}^\vee(\psi, \epsilon) = \mathit{dnf}(\psi)$, $\mathit{af}^\vee(\psi, \nu) = \mathit{dnf}(\mathit{af}(\psi, \nu))$, and $\mathit{af}^\vee(\psi, \nu w) = \bigcup_{\psi'\in\mathit{af}^\vee(\psi,\nu)} \mathit{af}^\vee(\psi', w)$ for every formula ψ, letter ν, and word w.

Notice that $\mathit{dnf}(\mathbf{ff}) = \emptyset$ and $\mathit{dnf}(\mathbf{tt}) = \{\mathbf{tt}\}$. Since the automata defined below have sets of states of the form $\mathit{Reach}^\vee(\varphi)$, they have a state labeled by $\mathbf{tt}$, but no state labeled by $\mathbf{ff}$.

The proof of the next proposition follows immediately from the definitions.

Proposition 7.2. *Let* $\varphi \in \mu LTL$.

- *The following NBA over the alphabet* 2^{Ap} *recognizes* $L(\varphi)$*:*
$$\mathcal{A}^\varphi_\mu = (\,\mathit{Reach}^\vee(\varphi), \mathit{af}^\vee, \mathit{dnf}(\varphi), \mathit{inf}(\mathbf{tt})\,)$$
- *The following NBA over the alphabet* 2^{Ap} *recognizes* $L(\mathbf{GF}\varphi)$*:*
$$\mathcal{A}^\varphi_{\mathbf{GF}\mu} = (\,\mathit{Reach}^\vee(\mathbf{F}\varphi), \mathit{af}^\vee_{\mathbf{F}\varphi}, \{\mathbf{F}\varphi\}, \mathit{inf}(\mathbf{tt})\,)$$
$$\mathit{af}^\vee_{\mathbf{F}\varphi}(\psi, \nu) = \begin{cases} \{\mathbf{F}\varphi\} & \text{if } \psi \equiv_P \mathbf{tt} \\ \mathit{af}^\vee(\psi, \nu) & \text{otherwise.} \end{cases}$$

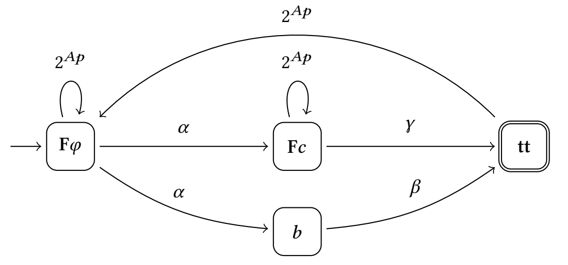
Let $\varphi \in \nu LTL$.

- *The following NBA over the alphabet* 2^{Ap} *recognizes* $L(\varphi)$*:*
$$\mathcal{A}^\varphi_\nu = (\,\mathit{Reach}^\vee(\varphi), \mathit{af}^\vee, \mathit{dnf}(\varphi), \mathit{inf}(\mathit{Reach}^\vee(\varphi))\,)$$
- *The following NBA over the alphabet* 2^{Ap} *recognizes* $L(\mathbf{FG}\varphi)$*:*
$$\mathcal{A}^\varphi_{\mathbf{FG}\nu} = (\,\mathit{Reach}^\vee(\mathbf{G}\varphi) \cup \{\mathbf{FG}\varphi\}, \mathit{af}^\vee_{\mathbf{G}\varphi}, \{\mathbf{FG}\varphi\}, \mathit{inf}(\mathit{Reach}^\vee(\mathbf{G}\varphi))\,)$$
$$\mathit{af}^\vee_{\mathbf{G}\varphi}(\psi, \nu) = \begin{cases} \{\mathbf{FG}\varphi, \mathbf{G}\varphi\} & \text{if } \psi = \mathbf{FG}\varphi \\ \mathit{af}^\vee(\psi, \nu) & \text{otherwise.} \end{cases}$$

Recall that the elements of $\mathit{Reach}(\varphi)$ are positive boolean combinations of proper subformulas of φ. It follows that the elements of $\mathit{Reach}^\vee(\varphi)$ are *conjunctions* of proper subformulas of φ. Since the number of proper subformulas is bounded by the length of the formula, we immediately obtain:

Proposition 7.3. *If* φ *has n proper subformulas, then* $\mathit{Reach}^\vee(\varphi)$ *has at most* 2^n *elements, and so all the NBAs of Proposition 7.2 have at most* $2^{n+1} + 1 = O(2^n)$ *states.*

Example 7.4. Let $\varphi = a \wedge \mathbf{X}(b \vee \mathbf{F}c)$, the formula for which a DBA was given in Example 4.3. The NBA $\mathcal{A}^{\varphi}_{\mathbf{GF}\mu}$ is shown below. The figure uses the abbreviations of Example 4.3.


$$\begin{tikzcd}
\arrow[r] & \mathbf{F}\varphi \arrow[loop above, "2^{Ap}"] \arrow[r, "\alpha"] \arrow[dr, "\alpha"'] & \mathbf{F}c \arrow[loop above, "2^{Ap}"] \arrow[r, "\gamma"] & \boxed{\mathbf{tt}} \arrow[ll, bend right, "2^{Ap}"'] \\
& & b \arrow[ur, "\beta"'] &
\end{tikzcd}$$

Compared to the DBA of Example 4.3, the NBA has a simpler structure, although in this case the same number of states. △

To define NBAs for arbitrary formulas we apply the Master Theorem. This is routine, and so we only sketch the constructions.

A NBA for $L^1_{X,Y}$. We define a NBA $C_{\varphi,X}$ that accepts a word w iff $w_i \models af(\varphi, w_{0i})[X]_\nu$ for some suffix w_i of w. Recall that $af(\varphi, w_{0i})[X]_\nu \in \nu LTL$ for every $i \geq 0$. The automaton consists of two components with sets of states Q_1 and Q_2 given by

$$Q_1 = \{(\psi, 1) \mid \psi \in Reach^\vee(\varphi)\} \quad Q_2 = \{(\psi[X]_\nu, 2) \mid \psi \in Reach^\vee(\varphi)\}$$

Transitions either stay in the same component, or "jump" from the first component to the second. Transitions that stay in the same component are of the form $(\psi, i) \xrightarrow{\nu} (\psi', i)$ for $\psi' \in af^\vee(\psi, \nu)$ and $i = 1, 2$. "Jumps" are transitions of the form $(\psi, 1) \xrightarrow{\epsilon} (\psi[X]_\nu, 2)$. Jumping amounts to nondeterministically guessing the suffix w_i satisfying $af(\varphi, w_{0i})[X]_\nu$. The accepting condition is $inf(Q_2)$. Notice that the state $(\mathbf{ff}, 2)$ does not have any successors.
Since $Reach^\vee(\varphi)$ has at most 2^n states, $C_{\varphi,X}$ has $2^{O(n)}$ states.

A NBA for $L^2_{X,Y}$. As in the case of DRAs, we define a NBA recognizing $L\left(\bigwedge_{\psi \in X} \mathbf{GF}(\psi[Y]_\mu)\right)$. To obtain an NBA with $2^{O(n)}$ states we use a well-known trick. Given a set $\{\psi_1, \ldots, \psi_k\}$ of formulas, we have

$$\bigwedge_{i=1}^{k} \mathbf{GF}\psi_i \equiv \mathbf{GF}(\psi_1 \wedge \mathbf{F}(\psi_2 \wedge \mathbf{F}(\psi_3 \wedge \ldots \wedge \mathbf{F}(\psi_{k-1} \wedge \mathbf{F}\psi_k) \ldots))$$

The formula obtained after applying the trick belongs to $\mathbf{GF}(\mu LTL)$ and has $O(n)$ μ-subformulas. By Proposition 7.2.2 we can construct a NBA for it with $2^{O(n)}$ states.

A NBA for $L^3_{X,Y}$. In this case we apply

$$\bigwedge_{i=1}^{k} \mathbf{FG}\psi_i \equiv \mathbf{FG}\left(\bigwedge_{i=1}^{k} \psi_i\right)$$

and Proposition 7.2.4, yielding an automaton with $2^{O(n)}$ states.

A NBA for $L(\varphi)$. We proceed as in the case of DRAs, using the well-known operations for union and intersection of NBAs. The NBA A_φ is the union of at most 2^n NBAs $A_{X,Y}$, each of them with $2^{O(n)}$ states. The difference with the DRA case is that, given NBAs with $n_1, \ldots, n_k$ states accepting languages $L_1, \ldots, L_k$, we can construct a NBA for $\bigcup_{i=1}^k L_i$ with $\sum_{i=1}^k n_i$ states, instead of $\prod_{i=1}^k n_i$ states, as was the case for DRAs. So A_φ has $2^n \cdot 2^{O(n)} = 2^{O(n)}$ states.

8 Constructing LDBAs for LTL Formulas

The translation of LTL into LDBA combines the translations into DRA and NBA. Recall that the states of an LDBA are partitioned into an initial component and a deterministic accepting component containing all accepting states. While in the definition of a LDBA the initial component can be nondeterministic, in our construction we can easily make it deterministic: Every accepting run has exactly one non-deterministic step. This makes the LDBA usable for quantitative (and not only qualitative) probabilistic model checking, as described in [35].

Lemma 6.1 shows that checking property (1) of Theorem 5.12 can be arbitrarily delayed, which allows us to slightly rephrase the Master Theorem as follows:

Theorem 8.1. *(Variant of the Master Theorem) For every formula φ and for every word w: $w \models \varphi$ iff there exists $X \subseteq \mu(\varphi)$, $Y \subseteq \nu(\varphi)$, and $i \geq 0$ satisfying*

$$\begin{array}{ll} (1') & w_i \models af(\varphi, w_{0i})[X]_\nu \\ (2') & \forall \psi \in X.\ w_i \models \mathbf{GF}(\psi[Y]_\mu) \\ (3') & \forall \psi \in Y.\ w_i \models \mathbf{G}(\psi[X]_\nu) \end{array}$$

Proof. Clearly, the existence of an index i satisfying (1'-3') implies that conditions (1-3) hold. For the other direction, assume conditions (1-3) hold. By Lemma 6.1 the index i of condition (1) can be chosen arbitrarily large. Since $w \models \bigwedge_{\psi \in X} \mathbf{FG}(\psi[X]_\nu)$, we can choose i so that it also satisfies $w_i \models \bigwedge_{\psi \in X} \mathbf{G}(\psi[X]_\nu)$. □

The idea of the construction is to use the initial component to keep track of $af(\varphi, w_{0i})$—that is, after reading a finite word w_{0i} the initial component is in state $af(\varphi, w_{0i})$—and use the jump to the accepting component to guess sets X and Y and the stabilization point i. The jump leads to the initial state of the intersection of three DBAs, which are in charge of checking (1′), (2′), and (3′).

Recall that $af(\varphi, w_{0i}) \in Reach(\varphi)$ for every word w and every $i \geq 0$. For every $\psi \in Reach(\varphi)$ and for each pair of sets X, Y we construct a DBA $\mathcal{D}_{\psi,X,Y}$ recognizing the intersection of the languages of the formulas:

$$\psi[X]_\nu \qquad \bigwedge_{\psi \in X} \mathbf{GF}(\psi[Y]_\mu) \qquad \bigwedge_{\psi \in Y} \mathbf{G}(\psi[X]_\nu)$$

These formulas belong to νLTL, $\mathbf{GF}(\mu LTL)$, and νLTL, respectively, and so we can obtain DBAs for them following the recipes of Proposition 4.2. As argued before, each of these DBAs have $2^{2^{O(n)}}$ states, and so we can also construct a DBA for their intersection with the same upper bound. Summarizing, we obtain:

Initial component. The component is $(Reach(\varphi), af, \{\varphi\})$ and thus the component has at most 2^{2^n} states. Recall that this component does not have accepting states.

Accepting component. The component is the disjoint union, for every $\psi \in Reach(\varphi)$, $X \subseteq \mu(\varphi)$, and $Y \subseteq \nu(\varphi)$, of the DBA $\mathcal{D}_{\psi,X,Y}$. Since $Reach(\varphi)$ has at most 2^{2^n} formulas and there are at most 2^n pairs (X, Y), the component is the disjoint union of at most $2^{2^n} \cdot 2^n$ automata, each of then with $2^{2^{O(n)}}$ states. Thus in total $2^{2^{O(n)}}$ states.

A LDBA for $L(\varphi)$. The LDBA is the disjoint union of the initial and accepting components. The initial component is connected to the accepting component by ϵ-transitions: For every formula $\psi \in Reach(\varphi)$ and for every two sets X, Y, there is an ϵ-transition from state ψ of the initial component to the initial state of $\mathcal{D}_{\psi,X,Y}$.

The LDBA has $2^{2^{O(n)}} + 2^{2^n} = 2^{2^{O(n)}}$ states. Recall that the lower bound for the blowup of a translation of LTL to LDBA is also doubly exponential (see e.g. [35]).

9 Discussion

This paper builds upon our own work [13, 15, 19, 26, 27, 35]. In particular, the notion of *stabilization point* of a word with respect to a formula, and the idea of using oracle information that is subsequently checked are already present there. The translations of LTL to LDBAs of [23, 24] are based on similar ideas, also with resemblance to obligation sets of [29, 30].

The essential novelty of this paper with respect to the previous work is the introduction of the symmetric mappings $\cdot[\cdot]_\mu$ and $\cdot[\cdot]_\nu$. Applying them to an arbitrary formula φ yields a simpler formula, but *not in the sense one might expect.* In particular, $\varphi[Y]_\mu$ may be *stronger* than φ. For example, the information that, say, the formula $a\mathbf{W}b$ does not hold infinitely often makes us check the *stronger* formula $a\mathbf{U}b = (a\mathbf{W}b)[\emptyset]_\mu$. However, exactly this point makes the "μ-ν-alternation" work: The formulas $\varphi[X]_\nu$ and $\varphi[Y]_\mu$ are only simpler in the sense of *easier to translate.* This is the reason why operators $\mathbf{W}$ and $\mathbf{M}$ are present in the core syntax and the missing piece since the symmetric solutions [26, 27], limited to fragments based on the simpler operators $\mathbf{F}$ and $\mathbf{G}$.

The Master Theorem can be applied beyond what is described in this paper. In order to translate LTL into universal automata we only need to normalize formulas into conjunctive normal form. Furthermore one can obtain a double exponential translation into deterministic parity automata adapting the approach described in [14]. Another intriguing question is whether our translation into NBA, which is very different from the ones described in the literature is of advantage in some application like runtime verification.

The target automata classes used in practice typically use an acceptance condition defined on transitions, instead of states. Further, they use *generalized* acceptance conditions, be it Büchi or Rabin. All our constructions can be restated effortlessly to yield automata with transition-based acceptance, and if generalized acceptance conditions are allowed then they become simpler and more succinct. The implementation used in our experiments actually uses these two features, which is described in the appendix of [16].

To conclude, in our opinion this paper successfully finishes the journey started in [26]. Via a single theorem it provides an arguably elegant (unified, symmetric, syntax-independent, not overly complex) and efficient (asymptotically optimal and practically relevant) translation of LTL into your favourite ω-automata.

Acknowledgments. The authors want to thank Alexandre Duret-Lutz, Benedikt Seidl, the anonymous reviewers, and the participants of the 2nd Jerusalem Winter School in Computer Science and Engineering on "Formal Verification" for their helpful comments and remarks.

References

[1] Valentin M. Antimirov. 1996. Partial Derivatives of Regular Expressions and Finite Automaton Constructions. *Theor. Comput. Sci.* 155, 2 (1996), 291–319.
[2] Tomáš Babiak, František Blahoudek, Alexandre Duret-Lutz, Joachim Klein, Jan Křetínský, David Müller, David Parker, and Jan Strejček. 2015. The Hanoi Omega-Automata Format. In *CAV*. 479–486.
[3] Tomás Babiak, Frantisek Blahoudek, Mojmír Křetínský, and Jan Strejcek. 2013. Effective Translation of LTL to Deterministic Rabin Automata: Beyond the (F, G)-Fragment. In *ATVA*. 24–39.
[4] Tomás Babiak, Mojmír Křetínský, Vojtech Rehák, and Jan Strejcek. 2012. LTL to Büchi Automata Translation: Fast and More Deterministic. In *TACAS*. 95–109.
[5] Christel Baier and Joost-Pieter Katoen. 2008. *Principles of Model Checking*. MIT Press.
[6] Christel Baier, Stefan Kiefer, Joachim Klein, Sascha Klüppelholz, David Müller, and James Worrell. 2016. Markov Chains and Unambiguous Büchi Automata. In *CAV*. 23–42.
[7] Janusz A. Brzozowski. 1964. Derivatives of Regular Expressions. *J. ACM* 11, 4 (1964), 481–494. https://doi.org/10.1145/321239.321249
[8] Krishnendu Chatterjee, Andreas Gaiser, and Jan Křetínský. 2013. Automata with Generalized Rabin Pairs for Probabilistic Model Checking and LTL Synthesis. In *CAV*. 559–575.
[9] Costas Courcoubetis and Mihalis Yannakakis. 1995. The Complexity of Probabilistic Verification. *J. ACM* 42, 4 (1995), 857–907.
[10] Jean-Michel Couvreur. 1999. On-the-Fly Verification of Linear Temporal Logic. In *FM*. 253–271.
[11] Marco Daniele, Fausto Giunchiglia, and Moshe Y. Vardi. 1999. Improved Automata Generation for Linear Temporal Logic. In *CAV*. 249–260.
[12] Alexandre Duret-Lutz, Alexandre Lewkowicz, Amaury Fauchille, Thibaud Michaud, Etienne Renault, and Laurent Xu. 2016. Spot 2.0 - A Framework for LTL and ω-Automata Manipulation. In *ATVA*. 122–129.
[13] Javier Esparza and Jan Křetínský. 2014. From LTL to Deterministic Automata: A Safraless Compositional Approach. In *CAV*. 192–208.
[14] Javier Esparza, Jan Křetínský, Jean-François Raskin, and Salomon Sickert. 2017. From LTL and Limit-Deterministic Büchi Automata to Deterministic Parity Automata. In *TACAS*. 426–442.
[15] Javier Esparza, Jan Křetínský, and Salomon Sickert. 2016. From LTL to deterministic automata - A safraless compositional approach. *Formal Methods in System Design* 49, 3 (2016), 219–271.
[16] Javier Esparza, Jan Křetínský, and Salomon Sickert. 2018. One Theorem to Rule Them All: A Unified Translation of LTL into ω-Automata. *CoRR* abs/1805.00748 (2018). arXiv:1805.00748 http://arxiv.org/abs/1805.00748
[17] Kousha Etessami and Gerard J. Holzmann. 2000. Optimizing Büchi Automata. In *CONCUR*. 153–167.
[18] Carsten Fritz. 2003. Constructing Büchi Automata from Linear Temporal Logic Using Simulation Relations for Alternating Büchi Automata. In *CIAA*. 35–48.
[19] Andreas Gaiser, Jan Křetínský, and Javier Esparza. 2012. Rabinizer: Small Deterministic Automata for LTL(F,G). In *ATVA*. 72–76.
[20] Paul Gastin and Denis Oddoux. 2001. Fast LTL to Büchi Automata Translation. In *CAV*. 53–65.
[21] Dimitra Giannakopoulou and Flavio Lerda. 2002. From States to Transitions: Improving Translation of LTL Formulae to Büchi Automata. In *FORTE*. 308–326.
[22] Ernst Moritz Hahn, Guangyuan Li, Sven Schewe, Andrea Turrini, and Lijun Zhang. 2015. Lazy Probabilistic Model Checking without Determinisation. In *CONCUR*. 354–367.
[23] Dileep Kini and Mahesh Viswanathan. 2015. Limit Deterministic and Probabilistic Automata for LTL $\setminus$ GU. In *TACAS*. 628–642.
[24] Dileep Kini and Mahesh Viswanathan. 2017. Optimal Translation of LTL to Limit Deterministic Automata. In *TACAS*. 113–129.
[25] Zuzana Komárková and Jan Křetínský. 2014. Rabinizer 3: Safraless Translation of LTL to Small Deterministic Automata. In *ATVA (LNCS)*, Vol. 8837. 235–241.
[26] Jan Křetínský and Javier Esparza. 2012. Deterministic Automata for the (F,G)-Fragment of LTL. In *CAV*. 7–22.
[27] Jan Křetínský and Ruslán Ledesma-Garza. 2013. Rabinizer 2: Small Deterministic Automata for LTL $\setminus$ GU. In *ATVA*. 446–450.
[28] Jan Křetínský, Tobias Meggendorfer, Clara Waldmann, and Maximilian Weininger. 2017. Index Appearance Record for Transforming Rabin Automata into Parity Automata. In *TACAS*. 443–460.
[29] Jianwen Li, Geguang Pu, Lijun Zhang, Zheng Wang, Jifeng He, and Kim Guldstrand Larsen. 2013. On the Relationship between LTL Normal Forms and Büchi Automata. In *Theories of Programming and Formal Methods - Essays Dedicated to Jifeng He on the Occasion of His 70th Birthday*. 256–270.
[30] Jianwen Li, Lijun Zhang, Shufang Zhu, Geguang Pu, Moshe Y. Vardi, and Jifeng He. 2018. An explicit transition system construction approach to LTL satisfiability checking. *Formal Asp. Comput.* 30, 2 (2018), 193–217.
[31] Nir Piterman. 2006. From Nondeterministic Buchi and Streett Automata to Deterministic Parity Automata. In *LICS*. 255–264.
[32] Amir Pnueli. 1977. The Temporal Logic of Programs. In *FOCS*. 46–57.
[33] Shmuel Safra. 1988. On the Complexity of omega-Automata. In *FOCS*. 319–327.
[34] Sven Schewe. 2009. Tighter Bounds for the Determinisation of Büchi Automata. In *FoSSaCS*. 167–181.
[35] Salomon Sickert, Javier Esparza, Stefan Jaax, and Jan Křetínský. 2016. Limit-Deterministic Büchi Automata for Linear Temporal Logic. In *CAV*. 312–332.
[36] Fabio Somenzi and Roderick Bloem. 2000. Efficient Büchi Automata from LTL Formulae. In *CAV*. 248–263.
[37] Moshe Y. Vardi. 1985. Automatic Verification of Probabilistic Concurrent Finite-State Programs. In *FOCS*. 327–338.
[38] Moshe Y. Vardi and Pierre Wolper. 1986. An Automata-Theoretic Approach to Automatic Program Verification (Preliminary Report). In *LICS*. 332–344.

Conditional Value-at-Risk for Reachability and Mean Payoff in Markov Decision Processes

Jan Křetínský
Institut für Informatik (I7)
Technische Universität München
Garching bei München, Bavaria, Germany
jan.kretinsky@in.tum.de

Tobias Meggendorfer
Institut für Informatik (I7)
Technische Universität München
Garching bei München, Bavaria, Germany
tobias.meggendorfer@in.tum.de

Abstract

We present the *conditional value-at-risk (CVaR)* in the context of Markov chains and Markov decision processes with reachability and mean-payoff objectives. CVaR quantifies risk by means of the expectation of the worst p-quantile. As such it can be used to design risk-averse systems. We consider not only CVaR constraints, but also introduce their conjunction with expectation constraints and quantile constraints (value-at-risk, VaR). We derive lower and upper bounds on the computational complexity of the respective decision problems and characterize the structure of the strategies in terms of memory and randomization.

CCS Concepts • **Theory of computation → Verification by model checking**;

ACM Reference Format:
Jan Křetínský and Tobias Meggendorfer. 2018. Conditional Value-at-Risk for Reachability and Mean Payoff in Markov Decision Processes. In *LICS '18: 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, July 9–12, 2018, Oxford, United Kingdom.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3209108.3209176

1 Introduction

Markov decision processes (MDP) are a standard formalism for modelling stochastic systems featuring non-determinism. The fundamental problem is to design a strategy resolving the non-deterministic choices so that the systems' behaviour is optimized with respect to a given objective function, or, in the case of multi-objective optimization, to obtain the desired trade-off. The objective function (in the optimization phrasing) or the query (in the decision-problem phrasing) consists of two parts. First, a payoff is a measurable function assigning an outcome to each run of the system. It can be real-valued, such as the *long-run average reward* (also called *mean payoff*), or a two-valued predicate, such as *reachability*. Second, the payoffs for single runs are combined into an overall outcome of the strategy, typically in terms of *expectation*. The resulting objective function is then for instance the expected long-run average reward, or the probability to reach a given target state.

Risk-averse control aims to overcome one of the main disadvantages of the expectation operator, namely its ignorance towards the incurred risks, intuitively phrased as a question *"How bad are the bad cases?"* While the standard deviation (or variance) quantifies the spread of the distribution, it does not focus on the bad cases and thus fails to capture the risk. There are a number of quantities used to deal with this issue:

- The *worst-case* analysis (in the financial context known as discounted maximum loss) looks at the payoff of the worst possible run. While this makes sense in a fully non-deterministic environment and lies at the heart of verification, in the probabilistic setting it is typically unreasonably pessimistic, taking into account events happening with probability 0, e.g., never tossing head on a fair coin.
- The *value-at-risk* (VaR) denotes the worst p-quantile for some $p \in [0, 1]$. For instance, the value at the 0.5-quantile is the median, the 0.05-quantile (the *vigintile* or *ventile*) is the value of the best run among the 5% worst ones. As such it captures the "reasonably possible" worst-case. See Fig. 1 for an example of VaR for two given probability density functions. There has been an extensive effort spent recently on the analysis of MDP with respect to VaR and the re-formulated notions of quantiles, percentiles, thresholds, satisfaction view etc., see below. Although VaR is more realistic, it tends to ignore outliers too much, as seen in Fig. 1 on the right. VaR has been characterized as *"seductive, but dangerous"* and *"not sufficient to control risk"* [8].
- The *conditional value-at-risk* (average value-at-risk, expected shortfall, expected tail loss) answers the question *"What to expect in the bad cases?"* It is defined as the expectation over all events worse than the value-at-risk, see Fig. 1. As such it describes the lossy tail, taking outliers into account, weighted respectively. In the degenerate cases, CVaR for $p = 1$ is the expectation and for $p = 0$ the (probabilistic) worst case. It is an established risk metric in finance, optimization and operations research, e.g. [1, 33], and *"is considered to be a more consistent measure of risk"* [33]. Recently, it started permeating to areas closer to verification, e.g. robotics [13].

Our contribution In this paper, we investigate optimization of MDP with respect to CVaR as well as the respective trade-offs with expectation and VaR. We study the VaR and CVaR operators for the first time with the payoff functions of weighted reachability and

LICS '18, July 9–12, 2018, Oxford, United Kingdom
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5583-4/18/07...$15.00
https://doi.org/10.1145/3209108.3209176

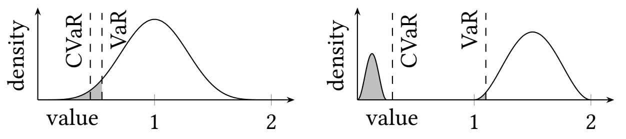


Figure 1. Illustration of VaR and CVaR for some random variables.

mean payoff, which are fundamental in verification. Moreover, we cover both the single-dimensional and the multi-dimensional case.

Particularly, we define CVaR for MDP and show the peculiarities of the concept. Then we study the computational complexity and the strategy complexity for various settings, proving the following:

- The single dimensional case can be solved in polynomial time through linear programming, see Section 5.
- The multi-dimensional case is NP-hard, even for CVaR-only constraints. Weighted reachability is NP-complete and we give PSPACE and EXPSPACE upper bounds for mean payoff with CVaR and expectation constraints, and with additional VaR constraints, respectively, see Section 6. (Note that already for the sole VaR constraints only an exponential algorithm is known; the complexity is an open question and not even NP-hardness is known [15, 32].)
- We characterize the strategy requirements, both in terms of memory, ranging from memoryless, over constant-size to infinite memory, and the required degree of randomization, ranging from fully deterministic strategies to randomizing strategies with stochastic memory update.

While dealing with the CVaR operator, we encountered surprising behaviour, preventing us to trivially adapt the solutions to the expectation and VaR problems:

- Compared to, e.g., expectation and VaR, CVaR does not behave linearly w.r.t. stochastic combination of strategies.
- A conjunction of CVaR constraints already is NP-hard, since it can force a strategy to play deterministically.

1.1 Related work

Worst case Risk-averse approaches optimizing the worst case together with expectation have been considered in beyond-worst-case and beyond-almost-sure analysis investigated in both the single-dimensional [11] and in the multi-dimensional [17] setup.

Quantiles The decision problem related to VaR has been phrased in probabilistic verification mostly in the form *"Is the probability that the payoff is higher than a given value threshold more than a given probability threshold?"* The total reward gained attention both in the verification community [6, 24, 35] and recently in the AI community [23, 29]. Multi-dimensional percentile queries are considered for various objectives, such as mean-payoff, limsup, liminf, shortest path in [32]; for the specifics of two-dimensional case and their interplay, see [3]. Quantile queries for more complex constraints have also been considered, namely their conjunctions [9, 20], conjunctions with expectations [15] or generally Boolean expressions [25]. Some of these approaches have already been practically applied and found useful by domain experts [4, 5].

CVaR There is a body of work that optimizes CVaR in MDP. However, to the best of our knowledge, all the approaches (1) focus on the single-dimensional case, (2) disregard the expectation, and (3) treat neither reachability nor mean payoff. They focus on the discounted [7], total [13], or immediate [27] reward, as well as extend the results to continuous-time models [26, 30]. This work comes from the area of optimization and operations research, with the notable exception of [13], which focuses on the total reward. Since the total reward generalizes weighted reachability, [13] is related to our work the most. However, it provides only an approximation solution for the one-dimensional case, neglecting expectation and the respective trade-offs.

Further, CVaR is a topic of high interest in finance, e.g., [8, 33]. The central difference is that there variations of portfolios (i.e. the objective functions) are considered while leaving the underlying random process (the market) unchanged. This is dual to our problem, since we fix the objective function and now search for an optimal random process (or the respective strategy).

Multi-objective expectation In the last decade, MDP have been extensively studied generally in the setting of multiple objectives, which provides some of the necessary tools for our trade-off analysis. Multiple objectives have been considered for both qualitative payoffs, such as reachability and LTL [19], as well as quantitative payoffs, such as mean payoff [9], discounted sum [14], or total reward [22]. Variance has been introduced to the landscape in [10].

2 Preliminaries

Due to space constraints, some proofs and explanations are shortened or omitted when clear and can be found in [28].

2.1 Basic definitions

We mostly follow the definitions of [9, 15]. $\mathbb{N}, \mathbb{Q}, \mathbb{R}$ are used to denote the sets of positive integers, rational and real numbers, respectively. For $n \in \mathbb{N}$, let $[n] = \{1, \dots, n\}$. Further, k_j refers to $k \cdot e_j$, where e_j is the unit vector in dimension j.

We assume familiarity with basic notions of probability theory, e.g., *probability space* $(\Omega, \mathcal{F}, \mu)$, *random variable* F, or *expected value* $\mathbb{E}$. The set of all distributions over a countable set C is denoted by $\mathcal{D}(C)$. Further, $d \in \mathcal{D}(C)$ is Dirac if $d(c) = 1$ for some $c \in C$. To ease notation, for functions yielding a distribution over some set C, we may write $f(\cdot, c)$ instead of $f(\cdot)(c)$ for $c \in C$.

Markov chains A *Markov chain* (MC) is a tuple $\mathsf{M} = (S, \delta, \mu_0)$, where S is a countable set of states[1], $\delta : S \to \mathcal{D}(S)$ is a probabilistic transition function, and $\mu_0 \in \mathcal{D}(S)$ is the initial probability distribution. The SCCs and BSCCs of a MC are denoted by SCC and BSCC, respectively [31].

A *run* in M is an infinite sequence $\rho = s_1 s_2 \cdots$ of states, we write ρ_i to refer to the i-th state s_i. A *path* ϱ in M is a finite prefix of a run ρ. Each path ϱ in M determines the set $\mathsf{Cone}(\varrho)$ consisting of all runs that start with ϱ. To M, we associate the usual probability space $(\Omega, \mathcal{F}, \mathbb{P})$, where Ω is the set of all runs in M, $\mathcal{F}$ is the σ-field generated by all $\mathsf{Cone}(\varrho)$, and $\mathbb{P}$ is the unique probability measure such that $\mathbb{P}(\mathsf{Cone}(s_1 \cdots s_k)) = \mu_0(s_1) \cdot \prod_{i=1}^{k-1} \delta(s_i, s_{i+1})$. Furthermore, $\Diamond B$ ($\Diamond\Box B$) denotes the set of runs which eventually reach (eventually remain in) the set $B \subseteq S$, i.e. all runs where $\rho_i \in B$ for some i (there exists an i_0 such that $\rho_i \in B$ for all $i \geq i_0$).

Markov decision processes A *Markov decision process* (MDP) is a tuple $\mathcal{M} = (S, \mathsf{A}, \mathsf{Av}, \Delta, s_0)$ where S is a finite set of states, A is a finite set of actions, $\mathsf{Av} : S \to 2^{\mathsf{A}} \setminus \{\emptyset\}$ assigns to each state s the set $\mathsf{Av}(s)$ of actions enabled in s so that $\{\mathsf{Av}(s) \mid s \in S\}$ is a partitioning of A[2], $\Delta : \mathsf{A} \to \mathcal{D}(S)$ is a probabilistic transition function that given an action a yields a probability distribution over the successor states, and s_0 is the initial state of the system.

[1]We allow the state set to be countable for the formal definition of strategies on MDP. When dealing with Markov Chains in queries, we only consider finite state sets.
[2]In other words, each action is associated with exactly one state.

A *run* ρ of $\mathcal{M}$ is an infinite alternating sequence of states and actions $\rho = s_1 a_1 s_2 a_2 \cdots$ such that for all $i \geq 1$, we have $a_i \in \mathsf{Av}(s_i)$ and $\Delta(a_i, s_{i+1}) > 0$. Again, ρ_i refers to the i-th state visited by this particular run. A *path* of length k in $\mathcal{M}$ is a finite prefix $\varrho = s_1 a_1 \cdots a_{k-1} s_k$ of a run in G.

Strategies and plays. Intuitively, a strategy in an MDP $\mathcal{M}$ is a "recipe" to choose actions based on the observed events. Usually, a strategy is defined as a function $\sigma : (S\mathsf{A})^* S \to \mathcal{D}(\mathsf{A})$ that given a finite path ϱ, representing the history of a play, gives a probability distribution over the actions enabled in the last state. We adopt the slightly different, though equivalent [9, Sec. 6] definition from [15], which is more convenient for our setting.

Let M be a countable set of *memory elements.* A *strategy* is a triple $\sigma = (\sigma_u, \sigma_n, \alpha)$, where $\sigma_u : \mathsf{A} \times S \times \mathsf{M} \to \mathcal{D}(\mathsf{M})$ and $\sigma_n : S \times \mathsf{M} \to \mathcal{D}(\mathsf{A})$ are *memory update* and *next move* functions, respectively, and $\alpha \in \mathcal{D}(\mathsf{M})$ is the initial memory distribution. We require that, for all $(s, m) \in S \times \mathsf{M}$, the distribution $\sigma_n(s, m)$ assigns positive values only to actions available at s, i.e. $\operatorname{supp} \sigma_n(s, m) \subseteq \mathsf{Av}(s)$.

A *play* of $\mathcal{M}$ determined by a strategy σ is a Markov chain $\mathcal{M}^\sigma = (S^\sigma, \delta^\sigma, \mu_0^\sigma)$, where the set of states is $S^\sigma = S \times \mathsf{M} \times \mathsf{A}$, the initial distribution μ_0 is zero except for $\mu_0^\sigma(s_0, m, a) = \alpha(m) \cdot \sigma_n(s_0, m, a)$, and the transition probability from $s^\sigma = (s, m, a)$ to $s'^\sigma = (s', m', a')$ is $\delta^\sigma(s^\sigma, s'^\sigma) = \Delta(a, s') \cdot \sigma_u(a, s', m, m') \cdot \sigma_n(s', m', a')$. Hence, $\mathcal{M}^\sigma$ starts in a location chosen randomly according to α and σ_n. In state (s, m, a) the next action to be performed is a, hence the probability of entering s' is $\Delta(a, s')$. The probability of updating the memory to m' is $\sigma_u(a, s', m, m')$, and the probability of selecting a' as the next action is $\sigma_n(s', m', a')$. Since these choices are independent, and thus we obtain the product above.

Technically, $\mathcal{M}^\sigma$ induces a probability measure $\mathbb{P}^\sigma$ on S^σ. Since we mostly work with the corresponding runs in the original MDP, we overload $\mathbb{P}^\sigma$ to also refer to the probability measure obtained by projecting onto S. Further, "almost surely" etc. refers to happening with probability 1 according to $\mathbb{P}^\sigma$. The expected value of a random variable $X : \Omega \to \mathbb{R}$ is $\mathbb{E}^\sigma[X] = \int_\Omega X \, d\mathbb{P}^\sigma$.

A convex combinations of two strategies σ_1 and σ_2, written as $\sigma_\lambda = \lambda\sigma_1 + (1 - \lambda)\sigma_2$, can be obtained by defining the memory as $\mathsf{M}_\lambda = \{1\} \times \mathsf{M}_1 \cup \{2\} \times \mathsf{M}_2$, randomly choosing one of the two strategies via the initial memory distribution α_λ and then following the chosen strategy. Clearly, we have that $\mathbb{P}^{\sigma_\lambda} = \lambda\mathbb{P}^{\sigma_1} + (1 - \lambda)\mathbb{P}^{\sigma_2}$.

Strategy types. A strategy σ may use infinite memory M, and both σ_u and σ_n may randomize. The strategy σ is

- *deterministic-update*, if α is Dirac and the memory update function σ_u gives a Dirac distribution for every argument;
- *deterministic*, if it is deterministic-update and the next move function σ_n gives a Dirac distribution for every argument.

A *stochastic-update* strategy is a strategy that is not necessarily deterministic-update and *randomized* strategy is a strategy that is not necessarily deterministic. We also classify the strategies according to the size of memory they use. Important subclasses are *memoryless* strategies, in which M is a singleton, *n-memory* strategies, in which M has exactly n elements, and *finite-memory* strategies, in which M is finite.

End components. A tuple (T, B) where $\emptyset \neq T \subseteq S$ and $\emptyset \neq B \subseteq \bigcup_{t \in T} \mathsf{Av}(t)$ is an *end component* of the MDP $\mathcal{M}$ if (i) for all actions $a \in B$, $\Delta(a, s') > 0$ implies $s' \in T$; and (ii) for all states $s, t \in T$ there is a path $\varrho = s_1 a_1 \cdots a_{k-1} s_k \in (TB)^{k-1} T$ with $s_1 = s$, $s_k = t$. An end component (T, B) is a *maximal end component (MEC)* if T and B are maximal with respect to subset ordering. Given an MDP, the set of MECs is denoted by MEC. By abuse of notation, $s \in M$ refers to all states of a MEC M, while $a \in M$ refers to the actions.

Remark 1. *Computing the maximal end component (MEC) decomposition of an MDP, i.e. the computation of* MEC*, is in P [18].*

Remark 2. *For any MDP $\mathcal{M}$ and strategy σ, a run almost surely eventually stays in one MEC, i.e. $\mathbb{P}^\sigma[\bigcup_{M_i \in \mathsf{MEC}} \Diamond\Box M_i] = 1$ [31].*

2.2 Random variables on Runs

We introduce two standard random variables, assigning a value to each run of a Markov Chain or Markov Decision Process.

Weighted reachability. Let $T \subseteq S$ be a set of target states and $\mathbf{r} : T \mapsto \mathbb{Q}$ be a reward function. Define the random variable R^{r} as $R^{\mathsf{r}}(\rho) = \mathbf{r}(\min_i\{\rho_i \mid \rho_i \in T\})$, if such an i exists, and 0 otherwise. Informally, R^{r} assigns to each run the value of the first visited target state, or 0 if none. R^{r} is measurable and discrete, as S is finite [31]. Whenever we are dealing with weighted reachability, we assume w.l.o.g. that all target states are absorbing, i.e. for any $s \in T$ we have $\delta(s, s) = 1$ for MC and $\Delta(a, s) = 1$ for all $a \in \mathsf{Av}(s)$ for MDP.

Mean payoff (also known as *long-run average reward*, and *limit average reward*). Again, let $\mathbf{r} : S \mapsto \mathbb{Q}$ be a reward function. The mean payoff of a run ρ is the average reward obtained per step, i.e. $R^{\mathsf{m}}(\rho) = \liminf_{n \to \infty} \frac{1}{n} \sum_{i=1}^{n} \mathbf{r}(\rho_i)$. The lim inf is necessary, since lim may not be defined in general. Further, R^{m} is measurable [31].

Remark 3. *There are several distinct definitions of "weighted reachability". The one chosen here primarily serves as foundation for the more general mean payoff.*

3 Introducing the Conditional Value-at-risk

In order to define our problem, we first introduce the general concept of *conditional value-at-risk* (CVaR), also known as *average value-at-risk*, *expected shortfall*, and *expected tail loss.* As already hinted, the CVaR of some real-valued random variable X and probability $p \in [0, 1]$ intuitively is the expectation below the worst p-quantile of X.

Let $X : \Omega \to \mathbb{R}$ be a random variable over the probability space $(\Omega, \mathcal{F}, \mathbb{P})$. The associated *cumulative density function* (CDF) $F_X : \mathbb{R} \to [0, 1]$ of X yields the probability of X being less than or equal to the given value r, i.e. $F_X(r) = \mathbb{P}(\{X(\omega) \leq r\})$. F is non-decreasing and right continuous with left limits (*càdlàg*).

The *value-at-risk* VaR_p is the worst p-quantile, i.e. a value v s.t. the probability of X attaining a value less than or equal to v is p:[3]

$$\mathrm{VaR}_p(X) := \sup\{r \in \mathbb{R} \mid F_X(r) \leq p\} \quad (\mathrm{VaR}_1(X) = \infty)$$

Then, with $v = \mathrm{VaR}_p(X)$, CVaR can be defined as [33]

$$\mathrm{CVaR}_p(X) := \mathbb{E}[X \mid X \leq v] = \frac{1}{p} \int_{(-\infty, v]} x \, dF_X,$$

with the corner cases $\mathrm{CVaR}_0 := \mathrm{VaR}_0$ and $\mathrm{CVaR}_1 = \mathbb{E}$.

Unfortunately, this definition only works as intended for continuous X, as shown by the following example.

[3]An often used, mostly equivalent definition is $\inf\{r \in \mathbb{R} \mid F_X(r) \geq p\}$. Unfortunately, this would lead to some complications later on. See [28, Sec. A.1] for details.

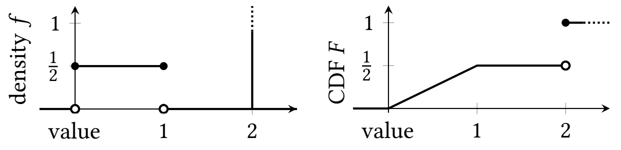


Figure 2. Distribution showing peculiarities of CVaR

Example 3.1. Consider a random variable X with a distribution as outlined in Fig. 2. For $p < \frac{1}{2}$, we certainly have $\mathrm{VaR}_p = 2p$. On the other hand, for any $p \in (\frac{1}{2}, 1)$, we get $\mathrm{VaR}_p = 2$. Consequently, the integral remains constant and CVaR_p would actually *decrease* for increasing p, not matching the intuition. △

General definition. As seen in Ex. 3.1, the previous definition breaks down when F_X is not continuous at the p-quantile and consequently $F_X(\mathrm{VaR}_p(X)) > p$. Thus, we handle the values at the threshold separately, similar to [34].

Definition 3.2. Let X be some random variable and $p \in [0, 1]$. With $v = \mathrm{VaR}_p(X)$, the CVaR of X is defined as

$$\mathrm{CVaR}_p(X) := \frac{1}{p}\left(\int_{(-\infty,v)} x\, dF_X + (p - \mathbb{P}[X < v]) \cdot v\right),$$

which can be rewritten as

$$\mathrm{CVaR}_p(X) = \tfrac{1}{p}\Big(\mathbb{P}[X < v] \cdot \mathbb{E}[X \mid X < v] + (p - \mathbb{P}[X < v]) \cdot v\Big).$$

The corner cases again are $\mathrm{CVaR}_0 := \mathrm{VaR}_0$, and $\mathrm{CVaR}_1 = \mathbb{E}$.

Since the degenerate cases of $p = 0$ and $p = 1$ reduce to already known problems, we exclude them in the following.

We demonstrate this definition on the previous example.

Example 3.3. Again, consider the random variable X from Ex. 3.1. For $\frac{1}{2} < p < 1$ we have that $\mathbb{P}[X < \mathrm{VaR}_p(X)] = \mathbb{P}[X < 2] = \frac{1}{2}$. The right hand side of the definition $(p - \mathbb{P}[X < \mathrm{VaR}_p(X)]) = p - \frac{1}{2}$ captures the remaining discrete probability mass which we have to handle separately. Together with $\int_{(-\infty,2)} x\, dF_X = \frac{1}{4}$ we get $\mathrm{CVaR}_p(X) = \frac{1}{p}(\frac{1}{4} + (p - \frac{1}{2}) \cdot 2) = 2 - \frac{3}{4p}$. For example, with $p = \frac{3}{4}$, this yields the expected result $\mathrm{CVaR}_p(X) = 1$. △

Remark 4. *Recall that* $\mathbb{P}[X < r]$ *can be expressed as the left limit of* F_X*, namely* $\mathbb{P}[X < r] = \lim_{r' \to^- r} F_X(r')$*. Hence,* $\mathrm{CVaR}_p(X)$ *solely depends on the CDF of* X *and thus random variables with the same CDF also have the same CVaR.*

We say that F_1 *stochastically dominates* F_2 for two CDF F_1 and F_2, if $F_1(r) \leq F_2(r)$ for all r. Intuitively, this means that a sample drawn from F_2 is likely to be larger or equal to a sample from F_1. All three investigated operators ($\mathbb{E}$, CVaR, and VaR) are monotone w.r.t. stochastic dominance [28, Sec. A.1].

4 CVaR in MC and MDP: Problem statement

Now, we are ready to define our problem framework. First, we explain the types of building blocks for our queries, namely lower bounds on expectation, CVaR, and VaR. Formally, we consider the following types of constraints.

$$\mathsf{e} \leq \mathbb{E}(X) \qquad \mathsf{c} \leq \mathrm{CVaR}_{\mathsf{p}}(X) \qquad \mathsf{v} \leq \mathrm{VaR}_{\mathsf{q}}(X)$$

X is some real-valued random variable, assigning a payoff to each run. With these constraints, the classes of queries are denoted by

$$\mathsf{MDP}^{\mathbf{crit}}_{\mathbf{obj},\mathbf{dim}}$$

- **crit** $\subseteq \{\mathbb{E}, \mathrm{CVaR}, \mathrm{VaR}\}$ are the types of constraints,
- **obj** $\in \{\mathsf{r}, \mathsf{m}\}$ is the type of the objective function, either weighted reachability r or mean payoff m, and
- **dim** $\in \{\mathsf{single}, \mathsf{multi}\}$ is the dimensionality of the query.

We use d to denote the dimensions of the problem, $d = 1$ iff **dim** = single. As usual, we assume that all quantities of the input, e.g., probabilities of distributions, are rational.

An instance of these queries is specified by an MDP $\mathcal{M}$, a d-dimensional reward function $\mathbf{r} : S \to \mathbb{Q}^d$, and constraints from **crit**, given by vectors $\mathbf{e}, \mathbf{c}, \mathbf{v} \in (\mathbb{Q} \cup \{\bot\})^d$ and $\mathbf{p}, \mathbf{q} \in (0, 1)^d$. This implies that in each dimension there is at most one constraint per type. The presented methods can easily be extended to the more general setting of multiple constraints of a particular type in one dimension. The decision problem is to determine whether there exists a strategy σ such that *all* constraints are met.

Technically, this is defined as follows. Let X be the d-dimensional random variable induced by the objective **obj** and reward function $\mathbf{r}$, operating on the probability space of $\mathcal{M}^\sigma$. The strategy σ is a witness to the query iff for each dimension $j \in [d]$ we have that $\mathbb{E}[X_j] \geq \mathbf{e}_j$, $\mathrm{CVaR}_{\mathbf{p}_j}(X_j) \geq \mathbf{c}_j$, and $\mathrm{VaR}_{\mathbf{q}_j}(X_j) \geq \mathbf{v}_j$. Moreover, $\bot$ constraints are trivially satisfied.

For completeness sake, we also consider $\mathsf{MC}^{\mathbf{crit}}_{\mathbf{obj},\mathbf{dim}}$ queries, i.e. the corresponding problem on (finite state) Markov chains.

Notation. We introduce the following abbreviations. When dealing with an MDP $\mathcal{M}$, CVaR^σ_p denotes CVaR_p relative to the probability space over runs induced by the strategy σ. When additionally the random variable X (e.g., mean payoff) is clear from the context, we may write CVaR_p and CVaR^σ_p instead of $\mathrm{CVaR}_p(X)$ and $\mathrm{CVaR}^\sigma_p(X)$, respectively. We also define analogous abbreviations for VaR.

5 Single dimension

We show that all queries in one dimension are in P. Furthermore, our LP-based decision procedures directly yield a description of a witness strategy and allow for optimization objectives. We refer to the input constraints by e for expectation, (p, c) for CVaR, and (q, v) for VaR. Further, we use i for indices related to SCCs / MECs.

5.1 Weighted reachability

First, we show the simple result for Markov Chains, providing some insight in the techniques used in the MDP case.

Theorem 5.1. $\mathsf{MC}^{\{\mathbb{E},\mathrm{CVaR},\mathrm{VaR}\}}_{\mathsf{r},\mathsf{single}}$ *is in P.*

Proof. Let M be a *finite-state* Markov chain, $\mathbf{r}$ a reward function, and $T = \{b_1, \ldots, b_n\}$ the target set. Recall that all b_i are absorbing, hence single-state BSCCs. We obtain the stationary distribution p of M in polynomial time by, e.g., solving a linear equation system [31]. With p, we can directly compute the CDF of R^{r} as $F_{R^{\mathsf{r}}}(v) = \sum_{b_i : \mathbf{r}(b_i) \leq v} p(b_i)$ and immediately decide the query. □

Let us consider the more complex case of MDP. We show a lower bound on the type of strategies necessary to realize **obj** = r queries with constraints on expectation and one of VaR or CVaR. We then continue to prove that this class of strategies is optimal. This characterization is used to derive a polynomial time decision procedure based on a linear program (LP) which immediately yields a witness strategy. Finally, when we deal with the mean payoff case in Sec. 5.2, we make use of the reasoning presented in this section.

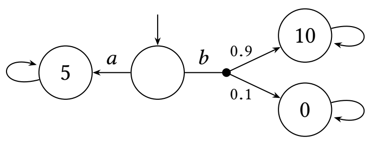


Figure 3. MDP used to show various difficulties of CVaR

Randomization is necessary for weighted reachability. In the following example, we present a simple MDP on which all deterministic strategies fail to satisfy specific constraints, while a straightforward randomizing one succeeds in doing so.

Example 5.2. Consider the MDP outlined in Fig. 3. The only non-determinism is given by the choice in the initial state s_0. Hence, any strategy is characterised by the choice in that particular state. Let now σ_a and σ_b denote the deterministic strategies playing a and b in s_0, respectively. Clearly, σ_a achieves an expectation, $\mathrm{CVaR}^{\sigma_a}_{0.05}$, and $\mathrm{VaR}^{\sigma_a}_{0.05}$ of 5. On the other hand, σ_b obtains an expectation of 9 with $\mathrm{CVaR}^{\sigma_b}_{0.05}$ and $\mathrm{VaR}^{\sigma_b}_{0.05}$ equal to 0.

Thus, neither strategy satisfies the constraints $\mathsf{q} = \mathsf{p} = 0.05$, $\mathsf{e} = 6$, and $\mathsf{c} = 2$ (or $\mathsf{v} = 5$). This is the case even when the strategy has arbitrary (deterministic) memory at its disposal, since in the first step there is nothing to remember. Yet, $\sigma = \frac{3}{4}\sigma_a + \frac{1}{4}\sigma_b$ achieves $\mathbb{E} = \frac{3}{4}5 + \frac{1}{4}9 = 6 \geq \mathsf{e}$, $\mathrm{CVaR}_{\mathsf{p}} = 2.5 \geq \mathsf{c}$, and $\mathrm{VaR}_{\mathsf{q}} = 5 \geq \mathsf{v}$. △

Hence strategies satisfying an expectation constraint together with either a CVaR *or* VaR constraint may necessarily involve randomization in general. We prove that (i) under mild assumptions randomization actually is sufficient, i.e. no memory is required, and (ii) fixed memory may additionally be required in general.

Definition 5.3. Let $\mathcal{M}$ be an MDP with target set T and reward function $\mathbf{r}$. We say that $\mathcal{M}$ satisfies the *attraction assumption* if **A1**) the target set T is reached almost surely for any strategy, or **A2**) for all target state $s \in T$ we have $\mathbf{r}(s) \geq 0$.

Essentially, this definition implies that an optimal strategy never remains in a non-target MEC. This allows us to design memoryless strategies for the weighted reachability problem.

Theorem 5.4. *Memoryless randomizing strategies are sufficient for* $\mathsf{MDP}^{\{\mathbb{E},\mathrm{VaR},\mathrm{CVaR}\}}_{\mathsf{r},\mathsf{single}}$ *under the attraction assumption.*

Proof. Fix an MDP $\mathcal{M}$ and reward function $\mathbf{r}$. We prove that for any strategy σ there exists a memoryless, randomizing strategy σ' achieving at least the expectation, VaR, and CVaR of σ.

All target states $t_i \in T$ form single-state MECs, as we assumed that all target states are absorbing. Consequently, σ naturally induces a distribution over these s_i. Now, we apply [19, Theorem 3.2] to obtain a strategy σ' with $\mathbb{P}^{\sigma'}[\Diamond s_i] \geq \mathbb{P}^{\sigma}[\Diamond s_i]$ for all i.

With **A1**), we have $\sum p_i = 1$ and thus $\mathbb{P}^{\sigma'}[\Diamond t_i] = \mathbb{P}^{\sigma}[\Diamond t_i]$. Hence, σ' obtains the same CDF for the weighted reachability objective. Under **A2**), the CDF F' of strategy σ' stochastically dominates the CDF F of the original strategy σ, concluding the proof. □

Theorem 5.5. *Two-memory stochastic strategies (i.e. with both randomization and stochastic update) are sufficient for* $\mathsf{MDP}^{\{\mathbb{E},\mathrm{VaR},\mathrm{CVaR}\}}_{\mathsf{r},\mathsf{single}}$.

The proof is a simple application of the following Thm. 5.10, as weighted reachability is a special case of mean payoff. Together with an example for the lower bound it can be found in [28, Sec. A.2].

(1) All variables y_a, x_s, $\underline{x}_s$ are non-negative.
(2) Transient flow for $s \in S$:
$$\mathbb{1}_{s_0}(s) + \sum\nolimits_{a \in \mathsf{A}} y_a \Delta(a, s) = \sum\nolimits_{a \in \mathsf{Av}(s)} y_a + x_s$$
(3) Switching to recurrent behaviour:
$$\sum\nolimits_{s \in T} x_s = 1$$
(4) VaR-consistent split:
$$\underline{x}_s = x_s \text{ for } s \in T_< \qquad \underline{x}_s \leq x_s \text{ for } s \in T_=$$
(5) Probability-consistent split:
$$\sum\nolimits_{s \in T_\leq} \underline{x}_s = \mathsf{p}$$
(6) CVaR and expectation satisfaction:
$$\sum\nolimits_{s \in T_\leq} \underline{x}_s \cdot \mathbf{r}(s) \geq \mathsf{p} \cdot \mathsf{c} \qquad \sum\nolimits_{s \in T} x_s \cdot \mathbf{r}(s) \geq \mathsf{e}$$

Figure 4. LP used to decide weighted reachability queries given a guess t of $\mathrm{VaR}_{\mathsf{p}}$. $T_\sim := \{s \in T \mid \mathbf{r}(s) \sim t\}$, $\sim \in \{<, =, \leq\}$.

Inspired by [15, Fig. 3], we use the optimality result from Thm. 5.4 to derive a decision procedure for weighted reachability queries under the attraction assumptions based on the LP in Fig. 4.

To simplify the LP, we make further assumptions – see [28, Sec. A.2] for details. First, all MECs, including non-target ones, consist of a single state. Second, all MECs from which T is not reachable are considered part of T and have $\mathbf{r} = 0$ (similar to the "cleaned-up MDP" from [19]). Finally, we assume that the quantile-probabilities are equal, i.e. $\mathsf{p} = \mathsf{q}$. The LP can easily be extended to account for different values by duplicating the $\underline{x}_s$ variables and adding according constraints.

The central idea is to characterize randomizing strategies by the "flow" they achieve. To this end, Equality (2) essentially models Kirchhoff's law, i.e. inflow and outflow of a state have to be equal. In particular, y_a expresses the transient flow of the strategy as the expected total number of uses of action a. Similarly, x_s models the recurrent flow, which under our absorption assumption equals the probability of reaching s. Equality (3) ensures that all transient behaviour eventually changes into recurrent one.

In order to deal with our query constraints, Constraints (4) and (5) extract the worst p fraction of the recurrent flow, ensuring that the $\mathrm{VaR}_{\mathsf{p}}$ is at least t. Note that equality is not guaranteed by the LP; if $\underline{x}_s = x_s$ for all $s \in T_\leq$, we have $\mathrm{VaR}_{\mathsf{p}} > t$. Finally, Inequality (6) enforces satisfaction of the constraints.

Theorem 5.6. *Let $\mathcal{M}$ be an MDP with target states T and reward function $\mathbf{r}$, satisfying the attraction assumption. Fix the constraint probability* $\mathsf{p} \in (0, 1)$ *and thresholds* $\mathsf{e}, \mathsf{c} \in \mathbb{Q}$. *Then, we have that*

1. *for any strategy σ satisfying the constraints, there is a $t \in \mathbf{r}(S)$ such that the LP in Fig. 4 is feasible, and*
2. *for any threshold $t \in \mathbf{r}(S)$, a solution of the LP in Fig. 4 induces a memoryless, randomizing strategy σ satisfying the constraints and* $\mathrm{VaR}^{\sigma}_{\mathsf{p}} \geq t$.

Proof. First, we prove for a strategy σ satisfying the constraints that there exists a $t \in \mathbf{r}(S)$ such that the LP is feasible. By Thm. 5.4, we may assume that σ is a memoryless randomizing strategy. From [19, Theorem 3.2], we get an assignment to the y_a's and x_s's satisfying Equalities (1), (2), and (3) such that $\mathbb{P}^{\sigma}[\Diamond s] = x_s$ for all target states

$s \in T$. Further, let $v = \mathrm{VaR}^{\sigma}_{\mathrm{p}}$ be the value-at-risk of the strategy. By definition of VaR, we have that $\mathbb{P}^{\sigma}[X < v] \leq p$.

Assume for now that $\mathbb{P}^{\sigma}[X < v] = p$, i.e. the probability of obtaining a value strictly smaller than v is exactly p. In this case, choose t to be the next smaller reward, i.e. $t = \max\{\mathbf{r}(s) < v\}$. We set $\underline{x}_s = x_s$ for all $s \in T_{\leq}$, satisfying Constraints (4) and (5).

Otherwise, we have $\mathbb{P}^{\sigma}[X < v] < p$. Now, some non-zero fraction of the probability mass at v contributes to the CVaR. Again, we set the values for $\underline{x}_s$ according to Constraint (4). The only degree of freedom are the values of $\underline{x}_s$ where $\mathbf{r}(s) = t$. There, we assign the values so that $\sum_{s \in T_=} \underline{x}_s = p - \sum_{s \in T_<} \underline{x}_s$, satisfying Equality (5).

It remains to check Inequality (6). For expectation, we have $\sum_{s \in T} x_s \cdot \mathbf{r}(s) = \sum_{s \in T} \mathbb{P}^{\sigma}[\Diamond s] \cdot \mathbf{r}(s) = \mathbb{E}^{\sigma}[R^{\mathrm{r}}] \geq \mathrm{e}$. For CVaR, notice that, due to the already proven Constraints (4) and (5), the side of Inequality (6) is equal to $\mathrm{CVaR}^{\sigma}_{\mathrm{p}}$ and thus at least c.

Second, we prove that a solution to the LP induces the desired strategy σ. Again by [19, Theorem 3.2], we get a memoryless randomizing strategy σ such that $\mathbb{P}^{\sigma}[\Diamond s] = x_s$ for all states $s \in T$. Then $\mathbb{E}^{\sigma}[R^{\mathrm{r}}] = \sum_{s \in T} \mathbb{P}^{\sigma}[\Diamond s] \cdot \mathbf{r}(s) = \sum_{s \in T} x_s \cdot \mathbf{r}(s) \geq \mathrm{e}$. Further,

$$\mathrm{CVaR}_p(R^{\mathrm{r}}) = \frac{1}{\mathrm{p}} \left(\sum\nolimits_{s:\mathbf{r}(s)<v} x_s \cdot \mathbf{r}(s) + (\mathrm{p} - \sum\nolimits_{s:\mathbf{r}(s)<v} x_s) \cdot v \right)$$

by definition. Now, we make a case distinction on $\underline{x}_s = x_s$ for all $s \in T_=$. If this is true, we have $v = \mathrm{VaR}^{\sigma}_{\mathrm{p}} = \min\{r \in \mathbf{r}(S) \mid r > t\}$, but $\mathbb{P}^{\sigma}[X < v] = p$. Consequently, $T_{\leq} = \{s \in T : \mathbf{r}(s) < v\}$ and $\sum_{s:\mathbf{r}(s)<v} x_s = p$. Otherwise, we have $v = t$ and consequently $T_< = \{s \mid \mathbf{r}(s) < v\}$. Inserting in the above equation immediately gives the result $\mathrm{CVaR}_p(R^{\mathrm{r}}) = \frac{1}{\mathrm{p}} \sum_{s \in T_{\leq}} \mathbf{r}(s) \cdot \underline{x}_s$. □

The linear program requires to know the $\mathrm{VaR}^{\sigma}_{p}$ beforehand, which in turn clearly depends on the chosen strategy. Yet, there are only linearly many values the random variable R^{r} attains. Thus we can simply try to find a solution for all potential values of $\mathrm{VaR}^{\sigma}_{p}$, i.e. $\{r \in \mathbf{r}(S)\}$, yielding a polynomial time solution.

Corollary 5.7. $\mathsf{MDP}^{\{\mathbb{E}, \mathrm{VaR}, \mathrm{CVaR}\}}_{\mathrm{r}, \mathrm{single}}$ *is in P.*

Proof. Under the attraction assumption, this follows directly from Thm. 5.6. In general, the reduction to mean payoff used by Thm. 5.5 and the respective result from Cor. 5.11 show the result. □

5.2 Mean payoff

In this section, we investigate the case of **obj** = m. Again, the construction for MC is considerably simple, yet instructive for the following MDP case.

Theorem 5.8. $\mathsf{MC}^{\{\mathbb{E}, \mathrm{VaR}, \mathrm{CVaR}\}}_{\mathrm{m}, \mathrm{single}}$ *is in P.*

Proof sketch. For each BSCC B_i, we obtain its expected mean payoff $r_i = \mathbb{E}[R^{\mathrm{m}} \mid B_i]$ through, e.g., a linear equation system [31]. Almost all runs in B_i achieve this mean payoff and thus the corresponding random variable is discrete. We reduce the problem to weighted reachability by using the known reformulation

$$\mathbb{P}[R^{\mathrm{m}} = c] = \sum\nolimits_{B_i : r_i = c} P[\Diamond B_i].$$

We replace each of these BSCCs by a representative b_i to obtain M'. Define the set of target states $T = \{b_i\}$ and the reachability reward function $\mathbf{r}'(b_i) = r_i$. By applying the approach of Thm. 5.1, we obtain the expectation, VaR, and CVaR for reachability in M' which by construction coincides with the respective values for mean payoff in M. □

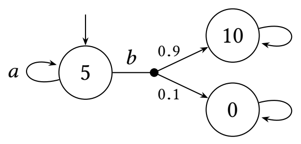


Figure 5. Memory is necessary for mean payoff queries

For the MDP case, recall that simple expectation maximization of mean payoff can be reduced to weighted reachability [2] and deterministic, memoryless strategies are optimal [31]. Yet, solving a conjunctive query involving either VaR or CVaR needs more powerful strategies than in the weighted reachability case of Thm. 5.4. Nevertheless, we show how to decide these queries in P.

Randomization and memory is necessary for mean payoff. A simple modification of the MDP in Fig. 3 yields an MDP where both randomization and memory is required to satisfy the constraints of the following example.

Example 5.9. Consider the MDP presented in Fig. 5. There, the same constraints as before, i.e. q = p = 0.05, e = 6, and c = 2 (or v = 5), can only be satisfied by strategies with both memory and randomization. Clearly, a pure strategy can only satisfy either of the two constraints again. But now a memoryless randomizing strategy also is insufficient, too, since any non-zero probability on action b leads to almost all runs ending up on the right side of the MDP, hence yielding a $\mathrm{CVaR}_{\mathrm{p}}$ and $\mathrm{VaR}_{\mathrm{q}}$ of 0. Instead, a stochastic strategy with $\mathsf{M} = \{a, b\}$ can simply choose $\alpha = \{a \mapsto \frac{3}{4}, b \mapsto \frac{1}{4}\}$ and play the corresponding action indefinitely, satisfying the constraints. △

We prove that this bound actually is tight, i.e. that, given stochastic memory update, two memory elements are sufficient.

Theorem 5.10. *Two-memory stochastic strategies (i.e. with both randomization and stochastic update) are sufficient for* $\mathsf{MDP}^{\{\mathbb{E}, \mathrm{VaR}, \mathrm{CVaR}\}}_{\mathrm{m}, \mathrm{single}}$.

Proof. Let σ be a strategy on an MDP $\mathcal{M}$ with reward function $\mathbf{r}$. We construct a two-memory stochastic strategy σ' achieving at least the expectation, VaR, and CVaR of σ.

First, we obtain a memoryless deterministic strategy σ_{opt} which obtains the maximal possible mean payoff in each MEC [31]. We then apply the construction of [9, Proposition 4.2] (see also [15, Lemma 5.7]), where the ξ is our σ_{opt}. (Technically, this can be ensured by choosing the constraints of the LP L according to σ_{opt}.)

Intuitively, this constructs a two-memory strategy σ' on $\mathcal{M}$ behaving as follows. Initially, σ' remains in each MEC with the same probability as σ, i.e. $\mathbb{P}^{\sigma'}[\Diamond\Box M_i] = \mathbb{P}^{\sigma}[\Diamond\Box M_i]$ by following a memoryless "searching" strategy and stochastically switching its memory state to "remain". Once in the "remain" state, the behaviour of the optimal strategy σ_{opt} is implemented.

Clearly, (i) both strategies remain in a particular MEC with the same probability, and (ii) σ' obtains as least as much value in each MEC as σ. Hence the CDF induced by σ' stochastically dominates the one of σ, concluding the proof. □

This immediately gives us a polynomial time decision procedure.

Corollary 5.11. $\mathsf{MDP}^{\{\mathbb{E}, \mathrm{VaR}, \mathrm{CVaR}\}}_{\mathrm{m}, \mathrm{single}}$ *is in P.*

Furthermore, we can use results of [15, Lemma 16] to trade the stochastic update for more memory.

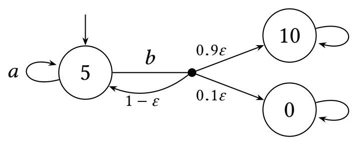


Figure 6. Exponential memory is necessary for mean payoff when only deterministic update is allowed.

Corollary 5.12. *Stochastic strategies with finite, deterministic memory are sufficient for* $\mathsf{MDP}^{\{\mathbb{E},\mathrm{VaR},\mathrm{CVaR}\}}_{\mathsf{m},\mathsf{single}}$.

Deterministic strategies may require exponential memory. As sources of randomness are not always available, one might ask what can be hoped for when only determinism is allowed. As already shown in Ex. 5.2, randomization is required in general. But even if some deterministic strategy is sufficient, it may require memory exponential in the size of the input, even in an MDP with only 3 states. We show this in the following example.

Example 5.13. Consider the MDP outlined in Fig. 6 together with the constraints $\mathsf{q} = \mathsf{p} = 0.05$, $\mathsf{e} = 6$, and $\mathsf{c} = 2$ (or $\mathsf{v} = 5$). Again, any optimal strategy needs a significant part of runs to go to the right side in order to satisfy the expectation constraint. Yet, any strategy can only "move" a small fraction of the runs there in each step. In particular, after k steps, the right side is only reached with probability at most $1 - (1 - \varepsilon)^k$. When choosing $\varepsilon = 2^{-n}$, which needs $\Theta(n)$ bits to encode, a deterministic strategy requires $k \geq c/\log(1-2^{-n}) \in O(2^n)$ memory elements to count the number of steps. The same holds true for any deterministic-update strategy.

On the other hand, a strategy with stochastic memory update can encode this counting by switching its state with a small probability after each step. For example, a strategy switching with probability $p = 3\varepsilon$ from "play b" to "play a" satisfies the constraint. △

5.3 Single constraint queries

In this section, we discuss an important sub-case of the single-dimensional case, namely queries with only a single constraint, i.e. $|\mathbf{crit}| = 1$. We show that deterministic memoryless strategies are sufficient in this case.

One might be tempted to use standard arguments and directly conclude this from the results of Thm. 5.4 as follows. Recall that this theorem shows that memoryless, randomizing strategies are sufficient; and that any such strategy can be written as finite convex combination of memoryless, deterministic strategies. Most constraints, for example expectation or reachability, behave linearly under convex combination of strategies, e.g., $\mathbb{E}^{\sigma_\lambda}(X) = \lambda\mathbb{E}^{\sigma_1}[X] + (1-\lambda)\mathbb{E}^{\sigma_2}[X]$. Consequently, for an optimal memoryless strategy, there is a deterministic witness, which in turn also is optimal.

Surprisingly, this assumption is not true for CVaR. On the contrary, the CVaR of a convex combination of strategies might be strictly worse than the CVaRs of either strategy, as shown in the following example. We prove a slightly weaker property of CVaR which eventually allows us to apply similar reasoning.

Example 5.14. Recall the MDP in Fig. 3 and let $p = 0.05$. As previously shown, $\mathrm{CVaR}_p^{\sigma_a} = 5$ and $\mathrm{CVaR}_p^{\sigma_b} = 0$, but the mixed strategy $\sigma_\lambda = \frac{1}{2}\sigma_a + \frac{1}{2}\sigma_b$ achieves $\mathrm{CVaR}_p^{\sigma_\lambda} = 0$ instead of the convex combination $\frac{1}{2}5 + \frac{1}{2}0 = 2.5$.

For $p = 0.2$, we have $\mathrm{CVaR}_p^{\sigma_a} = \mathrm{CVaR}_p^{\sigma_b} = 5$. Yet, *any* non-trivial convex combination of the two strategies yields a CVaR_p less than 5. See [28, Sec. A.1] for more details. With according constraints, this effectively can force an optimal strategy to choose between a or b. This observation is further exploited in the NP-hardness proof of the multi-dimensional case in Sec. 6. △

Since CVaR considers the worst events, the CVaR of a combination intuitively cannot be better than the combination of the respective CVaRs. We prove this intuition in the general setting, where instead of a convex combination of strategies we consider a mixture of two random variables.

Lemma 5.15. $\mathrm{CVaR}_p(X)$ *is convex in* X *for fixed* $p \in (0,1)$*, i.e. for random variables* X_1, X_2 *and* $\lambda \in [0,1]$

$$\mathrm{CVaR}_p(\lambda X_1 + (1-\lambda)X_2) \leq \lambda\,\mathrm{CVaR}_p(X_1) + (1-\lambda)\,\mathrm{CVaR}_p(X_2).$$

The proof can be found in [28, Sec. A.1]. This result allows us to apply the ideas outlined in the beginning of the section.

Theorem 5.16. *For any* $\mathbf{obj} \in \{\mathsf{r},\mathsf{m}\}$*, deterministic memoryless strategies are sufficient for* $\mathsf{MDP}^{\mathbf{crit}}_{\mathbf{obj},\mathsf{single}}$ *when* $|\mathbf{crit}| = 1$.

Proof. This is known for $\mathbf{crit} = \{\mathbb{E}\}$ [31] and $\mathbf{crit} = \{\mathrm{VaR}\}$ [21].

For CVaR, observe that the convex combination of deterministic strategies cannot achieve a better CVaR than the best strategy involved in the combination (see Lem. 5.15). This immediately yields the result for $\mathbf{obj} = \mathsf{r}$ through Thm. 5.4. For $\mathbf{obj} = \mathsf{m}$, we exploit the approach of Thm. 5.10. Recall that there we obtained a two-memory strategy σ'. Both randomization and stochastic update are used solely to distribute the runs over all MECs accordingly. By the above reasoning, for each MEC it is sufficient to either almost surely remain there or leave it. This behaviour can be implemented by a deterministic memoryless strategy on the original MDP. □

6 Multiple Dimensions

In this section, we deal with multi-dimensional queries. We continue to use i for indices related to MECs and further use j for dimension indices. First, we show that the Markov Chain case does not significantly change.

Theorem 6.1. *For any* $\mathbf{obj} \in \{\mathsf{r},\mathsf{m}\}$*,* $\mathsf{MC}^{\{\mathbb{E},\mathrm{VaR},\mathrm{CVaR}\}}_{\mathbf{obj},\mathsf{multi}}$ *is in P.*

Proof. Similarly to the single-dimensional case, we decide each constraint in each dimension separately, using our previous results. The query is satisfied iff each of the constraints is satisfied. □

6.1 NP-Hardness of reachability and mean payoff

For the MDP on the other hand, multiple dimensions add significant complexity. In the following, we show that already the weighted reachability problem with multiple dimensions and only CVaR constraints, i.e. $\mathsf{MDP}^{\{\mathrm{CVaR}\}}_{\mathsf{r},\mathsf{multi}}$, is NP-hard. This result directly transfers to mean payoff, i.e. $\mathbf{obj} = \mathsf{m}$. Recall that in contrast $\mathsf{MDP}^{\{\mathbb{E}\}}_{\mathsf{r},\mathsf{multi}}$ and even $\mathsf{MDP}^{\{\mathbb{E},\mathrm{VaR}_0\}}_{\mathsf{r},\mathsf{multi}}$, i.e. constraints on the expectation and ensuring that almost all runs achieve a given threshold, are in P [15].

Theorem 6.2. *For any* $\mathbf{obj} \in \{\mathsf{r},\mathsf{m}\}$*,* $\mathsf{MDP}^{\{\mathrm{CVaR}\}}_{\mathbf{obj},\mathsf{multi}}$ *is NP-hard (when the dimension* d *is a part of the input).*

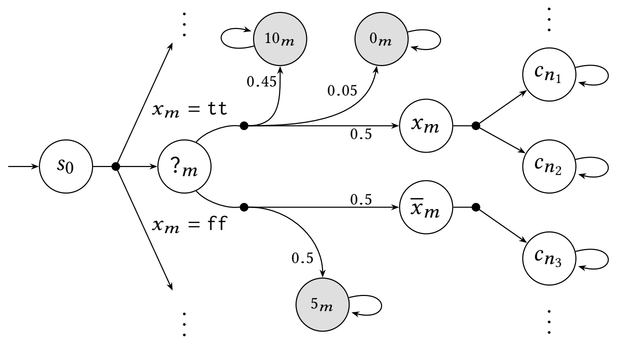


Figure 7. Gadget for variable x_m. Uniform transition probabilities are omitted for readability.

Proof. We prove hardness by reduction from 3-SAT. The core idea is to utilize observations from Fig. 3 and Ex. 5.14, namely that CVaR constraints can be used to enforce a deterministic choice.

Let $\{C_n\}$ be a set of N clauses with M variables x_m and set the dimensions $d = N+M$. By abuse of notation, n refers to the dimension of clause C_n and m to the one of variable x_m, respectively.

The gadget for the reduction is outlined in Fig. 7. Observe that, due to the structure of the MDP, we have that $R^{\mathsf{r}} = R^{\mathsf{m}}$.

Overall, the reduction works as follows. Initially, a state $?_m$, representing the variable x_m, is chosen uniformly. In this state, the strategy is asked to give the valuation of x_m through the actions "$x_m = \texttt{tt}$" or "$x_m = \texttt{ff}$". As seen in Ex. 5.14, the structure of the shaded states can be used to enforces a deterministic choice between the two actions. Particularly, in dimension m we require $\mathrm{CVaR}_p \geq 5$ for $p = \frac{M-1}{M} + \frac{1}{M} \cdot 0.5 \cdot 0.2$. Since all other gadgets yield 0 in dimension m and only half of the runs going through $?_m$ end up in the shaded area, this corresponds to Ex. 5.14, where $p = 0.2$.

Once in either state x_m or $\overline{x}_m$, a state c_n corresponding to a clause C_n satisfied by this assignment is chosen uniformly. In the example gadget, we would have $x_m \in C_{n_1} \cap C_{n_2}$, and $\overline{x}_m \in C_{n_3}$. We set the reward of c_n to 1_n. Then a clause c_n is satisfied under the assignment if the state c_n is visited with positive probability, e.g. if $\mathrm{CVaR}_1 \geq \frac{1}{M} \cdot 0.5 \cdot \frac{1}{N}$. Clearly, a satisfying assignment exists iff a strategy satisfying these constraints exists. ◻

6.2 NP-completeness and strategies for reachability

For weighted reachability, we prove that the previously presented bound is tight, i.e. that the weighted reachability problem with multiple dimensions and CVaR constraints is NP-complete when d is part of the input and P otherwise. First, we show that the strategy bounds of the single dimensional case directly transfer. Intuitively, this is the case since only the steady state distribution over the target set T is relevant, independent of the dimensionality.

Theorem 6.3. *Two-memory stochastic strategies (i.e. with both randomization and stochastic update) are sufficient for* $\mathsf{MDP}^{\{\mathbb{E},\mathrm{VaR},\mathrm{CVaR}\}}_{\mathsf{r},\mathsf{multi}}$. *Moreover, if* $\mathbf{r}_j(s) \geq 0$ *for all* $s \in T$ *and* $j \in [d]$, *then memoryless randomizing strategies are sufficient.*

Proof. Follows directly from the reasoning used in the proofs of Thm. 5.10 and Thm. 5.4. ◻

(1) All variables y_a, x_s, $\underline{x}^j_s$ are non-negative.

(4) VaR-consistent split for $j \in [d]$:

$$\underline{x}^j_s = x_s \text{ for } s \in T^j_{<} \qquad \underline{x}^j_s \leq x_s \text{ for } s \in T^j_{=}$$

(5) Probability-consistent split for $j \in [d]$:

$$\sum\nolimits_{s \in T^j_{\leq}} \underline{x}^j_s = \mathrm{p}_j$$

(6) CVaR and expectation satisfaction for $j \in [d]$:

$$\sum\nolimits_{s \in T^j_{\leq}} \underline{x}_s \cdot \mathbf{r}(s) \geq \mathrm{p}_j \cdot \mathbf{c}_j \qquad \sum\nolimits_{s \in T} x_s \cdot \mathbf{r}_j(s) \geq \mathbf{e}_j$$

Figure 8. LP used to decide multi-dimensional weighted reachability queries given a guess $\mathbf{t}$ of $\mathrm{VaR}_{\mathrm{p}_j}$. Equalities (2) and (3) are as in Fig. 4, $T^j_{\sim} := \{s \in T \mid \mathbf{r}_j(s) \sim \mathbf{t}_j\}$, $\sim \in \{<, =, \leq\}$.

Theorem 6.4. $\mathsf{MDP}^{\{\mathbb{E},\mathrm{VaR},\mathrm{CVaR}\}}_{\mathsf{r},\mathsf{multi}}$ *is in NP if d is a part of the input; moreover, it is in P for any fixed d.*

Proof sketch. To prove containment, we guess the VaR threshold vector $\mathbf{t}$ out of the set of potential ones, namely $\{r \mid \exists i \in [d], s \in T.\mathbf{r}_i(s) = r\}^d$ and use an LP to verify the solution. We again assume that each MEC can reach the target set and is single-state, as we did for Fig. 4. The arguments used to resolve this assumption are still applicable in the multi-dimensional setting. The LP consists of the flow Equalities (2) and (3) from the LP in Fig. 4 together with the modified (In)Equalities (4)-(6) as shown in Fig. 8.

The difference is that we extract the worst fraction of the flow *in each dimension.* Consequently, we have d instances of each $\underline{x}_s$ variable, namely $\underline{x}^j_s$. The number of possible guesses $\mathbf{t}$ is bounded by $|T|^d$ and thus the guess is of polynomial length. For a fixed d the bound itself is polynomial and hence, as previously, we can try out all vectors. ◻

6.3 Upper bounds of mean payoff

In this section, we provide an upper bound on the complexity of mean-payoff queries. Strategies in this context are known to have higher complexity.

Proposition 6.5 ([9]). *Infinite memory is necessary for* $\mathsf{MDP}^{\{\mathbb{E}\}}_{\mathsf{m},\mathsf{multi}}$.

Note that this directly transfers to $\mathsf{MDP}^{\{\mathrm{CVaR}\}}_{\mathsf{m},\mathsf{multi}}$, as $\mathrm{CVaR}_1 = \mathbb{E}$. However, closing gaps between lower and upper bounds for the mean payoff objective is notoriously more difficult. For instance, $\mathsf{MDP}^{\{\mathrm{VaR}\}}_{\mathsf{m},\mathsf{multi}}$ is known to be in EXP, but not even known to be NP-hard, neither is $\mathsf{MDP}^{\{\mathbb{E},\mathrm{VaR}\}}_{\mathsf{m},\mathsf{multi}}$. Since we have proven that $\mathsf{MDP}^{\{\mathrm{CVaR}\}}_{\mathsf{m},\mathsf{multi}}$ is NP-hard, we can expect that obtaining the matching NP upper bound will be yet more difficult. The fundamental difference of the multi-dimensional mean-payoff case is that the solutions within MECs cannot be pre-computed, rather non-trivial trade-offs must be considered. Moreover, the trade-offs are not "local" and must be synchronized over all the MECs, see [15] for details.

We now observe that, as opposed to quantile queries, i.e. VaR constraints, the behaviour inside each MEC can be assumed to be quite simple. Our results primarily rely on [16] and use a similar notation. In particular, given a run ρ, $\mathrm{Freq}_a(\rho)$ yields the average frequency of action a, i.e. $\mathrm{Freq}_a(\rho) := \liminf_{n\to\infty} \frac{1}{n} \sum_{t=1}^{n} \mathbb{1}_a(a_t)$, where a_t refers to the action taken by ρ in step t.

Definition 6.6. A strategy σ is *MEC-constant* if for all $M_i \in \mathsf{MEC}$ with $\mathbb{P}^\sigma[\Diamond\Box M_i] > 0$ and all $j \in [d]$ there is a $v \in \mathbb{R}$ such that $\mathbb{P}^\sigma[R^{\mathsf{m}}_j = v \mid \Diamond\Box M_i] = 1$.

Lemma 6.7. *MEC-constant strategies are sufficient for* $\mathsf{MDP}^{\{\mathbb{E},\mathrm{CVaR}\}}_{\mathsf{m},\mathsf{multi}}$.

Proof. Fix an MDP $\mathcal{M}$ with MECs $\mathsf{MEC} = \{M_1, \dots, M_n\}$, reward function $\mathbf{r}$ and a strategy σ. Further, define $p_i = \mathbb{P}^\sigma[\Diamond\Box M_i]$. We construct a strategy σ' so that (i) $\mathbb{P}^{\sigma'}[\Diamond\Box M_i] = p_i$ for all M_i, and (ii) all behaviours of σ on a MEC M_i are "mixed" into each run on M_i, making it MEC-constant.

We first define the mixing strategies σ_i, achieving point (ii). By [16, Sec. 4.1], there are frequencies $(x_a)_{a\in\mathsf{A}}$ which

- satisfy $\sum_{a\in A} x_a \cdot \Delta(a,s) = \sum_{a\in\mathsf{Av}(s)} x_a$ for all $s \in S$,
- for each action a we have $\mathbb{E}^\sigma[\mathrm{Freq}_a] \le x_a$, and
- $\sum_{a\in\mathsf{A}\cap M_i} x_a = p_i$.

By [16, Cor. 5.5], there is a (Markov) strategy σ_i on M_i where

$$\mathbb{P}^{\sigma_i}\left[\mathrm{Freq}_a = x_a/p_i\right] = 1.$$

Consequently, σ_i is almost surely constant on M_i w.r.t. R^{m}. We apply the reasoning used in the proof of Thm. 5.10 to obtain the combined strategy σ' which achieves point (i) and switches to σ_i upon remaining in M_i.

Now, fix any $j \in [d]$, $M_i \in \mathsf{MEC}$, and $p, q \in (0,1)$. We have that $\mathbb{E}^{\sigma_i}[\mathrm{Freq}_a \mid \Diamond\Box M_i] \ge \mathbb{E}^\sigma[\mathrm{Freq}_a \mid \Diamond\Box M_i]$ by construction. Consequently, $\mathbb{E}^{\sigma'}(R^{\mathsf{m}}_j) \ge \mathbb{E}^\sigma(R^{\mathsf{m}}_j)$.

Since σ' is MEC-constant, we have $\mathrm{CVaR}^{\sigma'}_p(R^{\mathsf{m}}_j \mid \Diamond\Box M_i) = \mathbb{E}^{\sigma'}[R^{\mathsf{m}}_j \mid \Diamond\Box M_i]$. Further, by $\mathbb{E}^\sigma[\mathrm{Freq}_a \mid \Diamond\Box M_i] \cdot p_i \le \mathbb{E}^{\sigma_i}[\mathrm{Freq}_a]$ for all a, we get $\mathbb{E}^\sigma[R^{\mathsf{m}}_j \mid \Diamond\Box M_i] \le \mathbb{E}^{\sigma_i}[R^{\mathsf{m}}_j]$. So, $\mathrm{CVaR}^{\sigma_i}_p(R^{\mathsf{m}}_j) = \mathbb{E}^{\sigma_i}[R^{\mathsf{m}}_j] \ge \mathbb{E}^\sigma[R^{\mathsf{m}}_j \mid \Diamond\Box M_i] \ge \mathrm{CVaR}^\sigma_q(R^{\mathsf{m}}_j \mid \Diamond\Box M_i)$, as $\mathrm{CVaR} \le \mathbb{E}$.

Finally, we apply this inequality together with property (i), obtaining $\mathrm{CVaR}^\sigma_p(R^{\mathsf{m}}_j) \le \mathrm{CVaR}^{\sigma'}_p(R^{\mathsf{m}}_j)$ by [28, Thm. A.4] □

We utilize this structural property to design a linear program for these constraints. However, similarly to the previously considered LPs, it relies on knowing the VaR for each $\mathrm{CVaR}_{\mathbf{p}}$ constraint. Due to the non-linear behaviour of CVaR, the classical techniques do not allow us to conclude that VaR is polynomially sized and thus we do not present the "matching" NP upper bound, but a PSPACE upper bound, which we achieve as follows.

Theorem 6.8. $\mathsf{MDP}^{\{\mathbb{E},\mathrm{CVaR}\}}_{\mathsf{m},\mathsf{multi}}$ *is in PSPACE.*

Proof sketch. We use the existential theory of the reals, which is NP-hard and in PSPACE [12], to encode our problem. The VaR vector $\mathbf{t}$ is existentially quantified and the formula is a polynomially sized program with constraints linear in VaR's and linear in the remaining variables. This shows the complexity result.

The details of the procedure are as follows. For each $j \in [d]$, we use the existential theory of reals to guess the achieved VaR $\mathbf{t} = \mathrm{VaR}_{\mathbf{p}_j}$. Further, we non-deterministically obtain the following polynomially-sized information (or deterministically try out all options in PSPACE). For each $j \in [d]$ and for each MEC M_i, we guess if the value achieved in M_i is at most (denoted $M_i \in \mathsf{MEC}^j_\le$) or above (denoted $M_i \in \mathsf{MEC}^j_>$) the respective $\mathbf{t}_j$, and exactly one MEC $M^j_=$, which achieves a value equal to it. Given these guesses, we check whether the LP in Fig. 9 has a solution.

(1) All variables y_a, y_s, x_a, x_s are non-negative.

(2) Transient flow for $s \in S$:

$$\mathbb{1}_{s_0}(s) + \sum_{a\in\mathsf{A}} y_a \cdot \Delta(a,s) = \sum_{a\in\mathsf{Av}(s)} y_a + y_s$$

(3) Probability of switching in a MEC is the frequency of using its actions for $M_i \in \mathsf{MEC}$:

$$\sum_{s\in M_i} y_s = \sum_{a\in M_i} x_a$$

(4) Recurrent flow for $s \in S$:

$$x_s = \sum_{a\in\mathsf{A}} x_a \cdot \Delta(a,s) = \sum_{a\in\mathsf{Av}(s)} x_a$$

(5) CVaR and expectation satisfaction for $j \in [d]$:

$$\sum_{s\in S^j_\le} x_s \cdot \mathbf{r}_j(s) + \left(\mathbf{p}_j - \sum_{s\in S^j_\le} x_s\right) \cdot \mathbf{t}_j \ge \mathbf{p}_j \cdot \mathbf{c}_j$$

$$\sum_{s\in S} x_s \cdot \mathbf{r}_j(s) \ge \mathbf{e}_j$$

(6) Verify MEC classification guess for $j \in [d]$:

$$\sum_{s\in M^j_\le} x_s \cdot \mathbf{r}_j(s) \le \mathbf{t}_j \quad \text{for } M^j_\le \in \mathsf{MEC}^j_\le \cup \{M^j_=\}$$

$$\sum_{s\in M^j_\ge} x_s \cdot \mathbf{r}_j(s) \ge \mathbf{t}_j \quad \text{for } M^j_\ge \in \mathsf{MEC}^j_> \cup \{M^j_=\}$$

(7) Verify VaR guess for $j \in [d]$:

$$\sum_{s\in S^j_\le} x_s \le \mathbf{p}_j \qquad \sum_{s\in S^j_\le \cup M^j_=} x_s \ge \mathbf{p}_j$$

Figure 9. LP used to decide multi-dimensional mean-payoff queries given a guess $\mathbf{t}$ of $\mathrm{VaR}_{\mathbf{p}_j}$ and MEC classification $\mathsf{MEC}^j_\le$, $M^j_=$, and $\mathsf{MEC}^j_>$. $S^j_\sim := \{s \in S \mid s \in M \text{ and } M \in \mathsf{MEC}^j_\sim\}$, $\sim \in \{\le, >\}$.

Equations (1)-(4) describe the transient flow like the previous LP's and, additionally, the recurrent flow like in [31, Sec. 9.3] or [9, 16, 19]. This addition is needed, since now our MECs are not trivial, i.e. single state. Again, Inequalities (5) verify that the CVaR and expectation constraints are satisfied. Finally, Inequalities (6) and (7) verify the previously guessed information, i.e. the VaR vector and the MEC classification.

Using the very same techniques, it is easy to prove that solutions to the LP correspond to satisfying strategies and vice versa. In particular, Inequalities (6) and (7) directly make use of the MEC-constant property of Lem. 6.7. □

While MEC-constant strategies are sufficient for $\mathbb{E}$ with CVaR, in contrast, they are not even for just $\mathsf{MDP}^{\{\mathrm{VaR}\}}_{\mathsf{m},\mathsf{multi}}$ [15, Ex.22]. Consequently, only an exponentially large LP is known for $\mathsf{MDP}^{\{\mathrm{VaR}\}}_{\mathsf{m},\mathsf{multi}}$. We can combine all the objective functions together as follows:

Theorem 6.9. $\mathsf{MDP}^{\{\mathbb{E},\mathrm{VaR},\mathrm{CVaR}\}}_{\mathsf{m},\mathsf{multi}}$ *is in EXPSPACE.*

Proof sketch. We proceed exactly as in the previous case, but now the flows in Equality (4) are split into exponentially many flows, depending on the set of dimensions where they achieve the given VaR threshold, see LP L in [15, Fig. 4]. The resulting size of the program is polynomial in the size of the system and exponential in d. Hence the call to the decision procedure of the existential theory of reals results in the EXPSPACE upper bound. □

Table 1. Schematic summary of known and new results. Strategies are abbreviated by "C/n-M", where C is either *D*eterministic or *R*andomizing, n is the size of the memory, and M is either *D*etereministic or *S*tochastic *MEM*ory.

dim	single		multi				
obj	any		r	m			
crit	$\lvert\mathbf{crit}\rvert = 1$	$\lvert\mathbf{crit}\rvert \geq 2$	CVaR $\in$ **crit**	$\{\mathbb{E}, \text{VaR}_0\}$	$\{\text{VaR}\}$	$\{\text{CVaR}\}$, $\{\text{CVaR}, \mathbb{E}\}$	$\{\mathbb{E}, \text{CVaR}, \text{VaR}\}$
Complex.	P		NP-c., P for fixed d	P	EXP	NP-h., PSPACE	NP-h., EXPSPACE
Strat.	D/1-MEM	R/2-SMEM	R/2-SMEM	R/∞-DMEM			

7 Conclusion

We introduced the conditional value-at-risk for Markov decision processes in the setting of classical verification objectives of reachability and mean payoff. We observed that in the single dimensional case the additional CVaR constraints do not increase the computational complexity of the problems. As such they provide a useful means for designing risk-averse strategies, at no additional cost. In the multidimensional case, the problems become NP-hard. Nevertheless, this may not necessarily hinder the practical usability. Our results are summarized in Table 1.

We conjecture that the VaR's for given CVaR constraints are polynomially large numbers. In that case, the provided algorithms would yield NP-completeness for $\mathsf{MDP}^{\{\text{CVaR}\}}_{\mathsf{m},\mathsf{multi}}$ and EXPTIME-containment for $\mathsf{MDP}^{\{\mathbb{E},\text{VaR},\text{CVaR}\}}_{\mathsf{m},\mathsf{multi}}$, where the exponential dependency is only on the dimension, not the size of the system.

Acknowledgments

This research has been partially supported by the Czech Science Foundation grant No. 18-11193S and the German Research Foundation (DFG) project KR 4890/2 "Statistical Unbounded Verification" (383882557). We thank Vojtěch Forejt for bringing up the topic of CVaR and the initial discussions with Jan Krčál and wish them both happy life in industry. We also thank Michael Luttenberger and the anonymous reviewers for insightful comments and valuable suggestions.

References

[1] Philippe Artzner, Freddy Delbaen, Jean-Marc Eber, and David Heath. 1999. Coherent Measures of Risk. *Mathematical Finance* 9, 3 (1999), 203–228.

[2] Pranav Ashok, Krishnendu Chatterjee, Przemyslaw Daca, Jan Křetínský, and Tobias Meggendorfer. 2017. Value Iteration for Long-Run Average Reward in Markov Decision Processes. In *CAV (LNCS)*, Vol. 10426. Springer, 201–221.

[3] Christel Baier, Marcus Daum, Clemens Dubslaff, Joachim Klein, and Sascha Klüppelholz. 2014. Energy-Utility Quantiles. In *NFM (LNCS)*, Vol. 8430. Springer, 285–299.

[4] Christel Baier, Clemens Dubslaff, and Sascha Klüppelholz. 2014. Trade-off analysis meets probabilistic model checking. In *CSL-LICS*. ACM, 1:1–1:10.

[5] Christel Baier, Clemens Dubslaff, Sascha Klüppelholz, Marcus Daum, Joachim Klein, Steffen Märcker, and Sascha Wunderlich. 2014. Probabilistic Model Checking and Non-standard Multi-objective Reasoning. In *FASE (LNCS)*, Vol. 8411. Springer, 1–16.

[6] Christel Baier, Joachim Klein, Sascha Klüppelholz, and Sascha Wunderlich. 2017. Maximizing the Conditional Expected Reward for Reaching the Goal. In *TACAS (LNCS)*, Vol. 10206. 269–285.

[7] Nicole Bäuerle and Jonathan Ott. 2011. Markov Decision Processes with Average-Value-at-Risk criteria. *Math. Meth. of OR* 74, 3 (2011), 361–379.

[8] Tanya Styblo Beder. 1995. VAR: Seductive but dangerous. *Financial Analysts Journal* 51, 5 (1995), 12–24.

[9] Tomás Brázdil, Václav Brozek, Krishnendu Chatterjee, Vojtech Forejt, and Antonín Kucera. 2014. Two Views on Multiple Mean-Payoff Objectives in Markov Decision Processes. *LMCS* 10, 1 (2014).

[10] Tomás Brázdil, Krishnendu Chatterjee, Vojtech Forejt, and Antonín Kucera. 2013. Trading Performance for Stability in Markov Decision Processes. In *LICS*. IEEE Computer Society, 331–340.

[11] Véronique Bruyère, Emmanuel Filiot, Mickael Randour, and Jean-François Raskin. 2017. Meet your expectations with guarantees: Beyond worst-case synthesis in quantitative games. *Inf. Comput.* 254 (2017), 259–295.

[12] John F. Canny. 1988. Some Algebraic and Geometric Computations in PSPACE. In *STOC*. ACM, 460–467.

[13] Stefano Carpin, Yinlam Chow, and Marco Pavone. 2016. Risk aversion in finite Markov Decision Processes using total cost criteria and average value at risk. In *ICRA*. IEEE, 335–342.

[14] Krishnendu Chatterjee, Vojtech Forejt, and Dominik Wojtczak. 2013. Multi-objective Discounted Reward Verification in Graphs and MDPs. In *LPAR (LNCS)*, Vol. 8312. Springer, 228–242.

[15] Krishnendu Chatterjee, Zuzana Komárková, and Jan Křetínský. 2015. Unifying Two Views on Multiple Mean-Payoff Objectives in Markov Decision Processes. In *LICS*. IEEE Computer Society, 244–256.

[16] Krishnendu Chatterjee, Zuzana Křetínská, and Jan Křetínský. 2017. Unifying Two Views on Multiple Mean-Payoff Objectives in Markov Decision Processes. *LMCS* 13, 2 (2017).

[17] Lorenzo Clemente and Jean-François Raskin. 2015. Multidimensional beyond Worst-Case and Almost-Sure Problems for Mean-Payoff Objectives. In *LICS*. IEEE Computer Society, 257–268.

[18] Costas Courcoubetis and Mihalis Yannakakis. 1995. The Complexity of Probabilistic Verification. *J. ACM* 42, 4 (1995), 857–907.

[19] Kousha Etessami, Marta Z. Kwiatkowska, Moshe Y. Vardi, and Mihalis Yannakakis. 2008. Multi-Objective Model Checking of Markov Decision Processes. *LMCS* 4, 4 (2008).

[20] J.A. Filar, D. Krass, and K.W Ross. 1995. Percentile performance criteria for limiting average Markov decision processes. *IEEE Trans. Automat. Control* 40, 1 (Jan 1995), 2–10.

[21] Jerzy A. Filar, Dmitry Krass, and Keith W. Ross. 1995. Percentile performance criteria for limiting average Markov decision processes. *IEEE Trans. Automat. Control* 40 (1995), 2–10.

[22] Vojtech Forejt, Marta Z. Kwiatkowska, Gethin Norman, David Parker, and Hongyang Qu. 2011. Quantitative Multi-objective Verification for Probabilistic Systems. In *TACAS (LNCS)*, Vol. 6605. Springer, 112–127.

[23] Hugo Gilbert, Paul Weng, and Yan Xu. 2017. Optimizing Quantiles in Preference-Based Markov Decision Processes. In *AAAI*. AAAI Press, 3569–3575.

[24] Christoph Haase and Stefan Kiefer. 2015. The Odds of Staying on Budget. In *ICALP (LNCS)*, Vol. 9135. Springer, 234–246.

[25] Christoph Haase, Stefan Kiefer, and Markus Lohrey. 2017. Computing quantiles in Markov chains with multi-dimensional costs. In *LICS*. IEEE Computer Society, 1–12.

[26] Yonghui Huang and Xianping Guo. 2016. Minimum Average Value-at-Risk for Finite Horizon Semi-Markov Decision Processes in Continuous Time. *SIAM Journal on Optimization* 26, 1 (2016), 1–28.

[27] Masayuki Kageyama, Takayuki Fujii, Koji Kanefuji, and Hiroe Tsubaki. 2011. Conditional Value-at-Risk for Random Immediate Reward Variables in Markov Decision Processes. *American J. Computational Mathematics* 1, 3 (2011), 183–188.

[28] Jan Křetínský and Tobias Meggendorfer. 2018. *Conditional Value-at-Risk for Reachability and Mean Payoff in Markov Decision Processes.* Technical Report abs/1805.xxxxx. arXiv.org.

[29] Xiaocheng Li, Huaiyang Zhong, and Margaret L. Brandeau. 2017. Quantile Markov Decision Process. *CoRR* abs/1711.05788 (2017).

[30] Christopher W. Miller and Insoon Yang. 2017. Optimal Control of Conditional Value-at-Risk in Continuous Time. *SIAM J. Control and Optimization* 55, 2 (2017), 856–884.

[31] M. L. Puterman. 1994. *Markov Decision Processes.* J. Wiley and Sons.

[32] Mickael Randour, Jean-François Raskin, and Ocan Sankur. 2017. Percentile queries in multi-dimensional Markov decision processes. *FMSD* 50, 2-3 (2017), 207–248.

[33] R. Tyrrell Rockafellar and Stanislav Uryasev. 2000. Optimization of Conditional Value-at-Risk. *Journal of Risk* 2 (2000), 21–41.

[34] R Tyrrell Rockafellar and Stanislav Uryasev. 2002. Conditional value-at-risk for general loss distributions. *Journal of banking & finance* 26, 7 (2002), 1443–1471.

[35] Michael Ummels and Christel Baier. 2013. Computing Quantiles in Markov Reward Models. In *FoSSaCS (LNCS)*, Vol. 7794. Springer, 353–368.

Value Iteration for Simple Stochastic Games: Stopping Criterion and Learning Algorithm*

Edon Kelmendi, Julia Krämer, Jan Křetínský, and Maximilian Weininger

Technical University of Munich

Abstract. Simple stochastic games can be solved by value iteration (VI), which yields a sequence of under-approximations of the value of the game. This sequence is guaranteed to converge to the value only in the limit. Since no stopping criterion is known, this technique does not provide any guarantees on its results. We provide the first stopping criterion for VI on simple stochastic games. It is achieved by additionally computing a convergent sequence of *over-approximations* of the value, relying on an analysis of the game graph. Consequently, VI becomes an anytime algorithm returning the approximation of the value and the current error bound. As another consequence, we can provide a simulation-based asynchronous VI algorithm, which yields the same guarantees, but without necessarily exploring the whole game graph.

1 Introduction

Simple stochastic game (SG) [Con92] is a zero-sum two-player game played on a graph by Maximizer and Minimizer, who choose actions in their respective vertices (also called states). Each action is associated with a probability distribution determining the next state to move to. The objective of Maximizer is to maximize the probability of reaching a given target state; the objective of Minimizer is the opposite.

Stochastic games constitute a fundamental problem for several reasons. From the theoretical point of view, the complexity of this problem[1] is known to be in $\mathbf{UP} \cap \mathbf{coUP}$ [HK66] , but no polynomial-time algorithm is known. Further, several other important problems can be reduced to SG, for instance parity games, mean-payoff games, discounted-payoff games and their stochastic extensions [CF11]. The task of solving SG is also polynomial-time equivalent to solving perfect information Shapley, Everett and Gillette games [AM09]. Besides, the problem is practically relevant in verification and synthesis. SG can model reactive systems, with players corresponding to the controller of the system and to its environment, where quantified uncertainty is explicitly modelled. This is useful in many application domains, ranging from smart energy management

* This research was funded in part by the German Excellence Initiative and the European Union Seventh Framework Programme under grant agreement No. 291763 for TUM – IAS, the Studienstiftung des deutschen Volkes project "Formal methods for analysis of attack-defence diagrams", the Czech Science Foundation grant No. 18-11193S, TUM IGSSE Grant 10.06 (PARSEC), and the German Research Foundation (DFG) project KR 4890/2-1 "Statistical Unbounded Verification".

[1] Formally, the problem is to decide, for a given $p \in [0, 1]$ whether Maximizer has a strategy ensuring probability at least p to reach the target.

[CFK$^+$13b] to autonomous urban driving [CKSW13], robot motion planning [LaV00] to self-adaptive systems [CMG14]; for various recent case studies, see e.g. [SK16]. Finally, since Markov decision processes (MDP) [Put14] are a special case with only one player, SG can serve as abstractions of large MDP [KKNP10].

Solution techniques There are several classes of algorithms for solving SG, most importantly strategy iteration (SI) algorithms [HK66] and value iteration (VI) algorithms [Con92]. Since the repetitive evaluation of strategies in SI is often slow in practice, VI is usually preferred, similarly to the special case of MDPs [KM17]. For instance, the most used probabilistic model checker PRISM [KNP11] and its branch PRISM-Games [CFK$^+$13a] use VI for MDP and SG as the default option, respectively. However, while SI is in principle a precise method, VI is an approximative method, which converges only in the limit. Unfortunately, there is no known stopping criterion for VI applied to SG. Consequently, there are no guarantees on the results returned in finite time. Therefore, current tools stop when the difference between the two most recent approximations is low, and thus may return arbitrarily imprecise results [HM17].

Value iteration with guarantees In the special case of MDP, in order to obtain bounds on the imprecision of the result, one can employ a *bounded* variant of VI [MLG05,BCC$^+$14] (also called *interval iteration* [HM17]). Here one computes not only an under-approximation, but also an over-approximation of the actual value as follows. On the one hand, iterative computation of the least fixpoint of Bellman equations yields an under-approximating sequence converging to the value. On the other hand, iterative computation of the greatest fixpoint yields an over-approximation, which, however, does not converge to the value. Moreover, it often results in the trivial bound of 1. A solution suggested for MDPs [BCC$^+$14,HM17] is to modify the underlying graph, namely to collapse end components. In the resulting MDP there is only one fixpoint, thus the least and greatest fixpoint coincide and both approximating sequences converge to the actual value. In contrast, for general SG no procedure where the greatest fixpoint converges to the value is known. In this paper we provide one, yielding a stopping criterion. We show that the pre-processing approach of collapsing is not applicable in general and provide a solution on the original graph. We also characterize SG where the fixpoints coincide and no processing is needed. The main technical challenge is that states in an end component in SG can have different values, in contrast to the case of MDP.

Practical efficiency using guarantees We further utilize the obtained guarantees to practically improve our algorithm. Similar to the MDP case [BCC$^+$14], the quantification of the error allows for ignoring parts of the state space, and thus a speed up without jeopardizing the correctness of the result. Indeed, we provide a technique where some states are not explored and processed at all, but their potential effect is still taken into accountThe information is further used to decide the states to be explored next and to be analyzed in more detail. To this end, simulations and learning are used as tools. While for MDP this idea has already demonstrated speed ups in orders of magnitude [BCC$^+$14,ACD$^+$17], this paper provides the first technique of this kind for SG.

Our contribution is summarized as follows

- We introduce a VI algorithm yielding both under- and over-approximation sequences, both of which converge to the value of the game. Thus we present the first stopping criterion for VI on SG and the first anytime algorithm with guaranteed precision. We also characterize when a simpler solution is sufficient.
- We provide a learning-based algorithm, which preserves the guarantees, but is in some cases more efficient since it avoids exploring the whole state space.
- We evaluate the running times of the algorithms experimentally, concluding that obtaining guarantees requires an overhead that is either negligible or mitigated by the learning-based approach.

Related work The works closest to ours are the following. As mentioned above, [BCC$^+$14,HM17] describe the solution to the special case of MDP. While [BCC$^+$14] also provides a learning-based algorithm, [HM17] discusses the convergence rate and the exact solution. The basic algorithm of [HM17] is implemented in PRISM [BKL$^+$17] and the learning approach of [BCC$^+$14] in Storm [DJKV17a]. The extension for SG where the interleaving of players is severely limited (every end component belongs to one player only) is discussed in [Ujm15].

Further, in the area of probabilistic planning, bounded real-time dynamic programming [MLG05] is related to our learning-based approach. However, it is limited to the setting of stopping MDP where the target sink or the non-target sink is reached almost surely under any pair of strategies and thus the fixpoints coincide. Our algorithm works for general SG, not only for stopping ones, without any blowup.

For SG, the tools implementing the standard SI and/or VI algorithms are PRISM-games [CFK$^+$13a], GAVS+ [CKLB11] and GIST [CHJR10]. The latter two are, however, neither maintained nor accessible via the links provided in their publications any more.

Apart from fundamental algorithms to solve SG, there are various practically efficient heuristics that, however, provide none or weak guarantees, often based on some form of learning [BT00,LL08,WT16,TT16,AY17,BBS08]. Finally, the only currently available way to obtain any guarantees through VI is to perform γ^2 iterations and then round to the nearest multiple of $1/\gamma$, yielding the value of the game with precision $1/\gamma$ [CH08]; here γ cannot be freely chosen, but it is a fixed number, exponential in the number of states and the used probability denominators. However, since the precision cannot be chosen and the number of iterations is always exponential, this approach is infeasible even for small games.

Organization of the paper Section 2 introduces the basic notions and revises value iteration. Section 3 explains the idea of our approach on an example. Section 4 provides a full technical treatment of the method as well as the learning-based variation. Section 5 discusses experimental results and Section 6 concludes. The appendix (available in [KKKW18]) gives technical details on the pseudocode as well as the conducted experiments and provides more extensive proofs to the theorems and lemmata; in this paper, there are only proof sketches and ideas.

2 Preliminaries

2.1 Basic definitions

A probability distribution on a finite set X is a mapping $\delta : X \to [0,1]$, such that $\sum_{x \in X} \delta(x) = 1$. The set of all probability distributions on X is denoted by $\mathcal{D}(X)$. Now we define stochastic games, in literature often referred as simple stochastic games or stochastic two-player games with a reachability objective.

Definition 1 (SG). *A* stochastic game (SG) *is a tuple* $(S, S_\Box, S_\bigcirc, \mathsf{s}_0, \mathsf{A}, \mathsf{Av}, \delta, \mathtt{1}, \mathtt{0})$, *where* S *is a finite set of* states *partitioned into the sets* $S_\Box$ *and* $S_\bigcirc$ *of states of the player* Maximizer *and* Minimizer*, respectively,* $\mathsf{s}_0, \mathtt{1}, \mathtt{0} \in S$ *is the* initial *state,* target *state, and* sink *state, respectively,* A *is a finite set of* actions, $\mathsf{Av} : S \to 2^{\mathsf{A}}$ *assigns to every state a set of* available *actions, and* $\delta : S \times \mathsf{A} \to \mathcal{D}(S)$ *is a* transition function *that given a state* s *and an action* $\mathsf{a} \in \mathsf{Av}(\mathsf{s})$ *yields a probability distribution over* successor *states.*

A Markov decision process (MDP) *is a special case of SG where* $S_\bigcirc = \emptyset$.

We assume that SGs are non-blocking, so for all states s we have $\mathsf{Av}(\mathsf{s}) \neq \emptyset$. Further, $\mathtt{1}$ and $\mathtt{0}$ only have one action and it is a self-loop with probability 1. Additionally, we can assume that the SG is preprocessed so that all states with no path to $\mathtt{1}$ are merged with $\mathtt{0}$.

For a state s and an available action $\mathsf{a} \in \mathsf{Av}(\mathsf{s})$, we denote the set of successors by $\mathsf{Post}(\mathsf{s}, \mathsf{a}) := \{\mathsf{s}' \mid \delta(\mathsf{s}, \mathsf{a}, \mathsf{s}') > 0\}$. Finally, for any set of states $T \subseteq S$, we use $T_\Box$ and $T_\bigcirc$ to denote the states in T that belong to Maximizer and Minimizer, whose states are drawn in the figures as $\Box$ and $\bigcirc$, respectively.

The semantics of SG is given in the usual way by means of strategies and the induced Markov chain and the respective probability space, as follows. An *infinite path* ρ is an infinite sequence $\rho = \mathsf{s}_0\mathsf{a}_0\mathsf{s}_1\mathsf{a}_1 \ldots \in (S \times \mathsf{A})^\omega$, such that for every $i \in \mathbb{N}$, $\mathsf{a}_i \in \mathsf{Av}(\mathsf{s}_i)$ and $\mathsf{s}_{i+1} \in \mathsf{Post}(\mathsf{s}_i, \mathsf{a}_i)$. *Finite path*s are defined analogously as elements of $(S \times \mathsf{A})^* \times S$. Since this paper deals with the reachability objective, we can restrict our attention to memoryless strategies, which are optimal for this objective. We still allow randomizing strategies, because they are needed for the learning-based algorithm later on. A *strategy* of Maximizer or Minimizer is a function $\sigma : S_\Box \to \mathcal{D}(\mathsf{A})$ or $S_\bigcirc \to \mathcal{D}(\mathsf{A})$, respectively, such that $\sigma(\mathsf{s}) \in \mathcal{D}(\mathsf{Av}(\mathsf{s}))$ for all s. We call a strategy *deterministic* if it maps to Dirac distributions only. Note that there are finitely many deterministic strategies. A pair (σ, τ) of strategies of Maximizer and Minimizer induces a Markov chain $\mathsf{G}^{\sigma,\tau}$ where the transition probabilities are defined as $\delta(\mathsf{s}, \mathsf{s}') = \sum_{\mathsf{a} \in \mathsf{Av}(\mathsf{s})} \sigma(\mathsf{s}, \mathsf{a}) \cdot \delta(\mathsf{s}, \mathsf{a}, \mathsf{s}')$ for states of Maximizer and analogously for states of Minimizer, with σ replaced by τ. The Markov chain induces a unique probability distribution $\mathbb{P}^{\sigma,\tau}_{\mathsf{s}}$ over measurable sets of infinite paths [BK08, Ch. 10].

We write $\Diamond\mathtt{1} := \{\rho \mid \exists i \in \mathbb{N}.\ \rho(i) = \mathtt{1}\}$ to denote the (measurable) set of all paths which eventually reach $\mathtt{1}$. For each $\mathsf{s} \in S$, we define the *value* in s as

$$\mathsf{V}(\mathsf{s}) := \sup_\sigma \inf_\tau \mathbb{P}^{\sigma,\tau}_s(\Diamond\mathtt{1}) = \inf_\tau \sup_\sigma \mathbb{P}^{\sigma,\tau}_s(\Diamond\mathtt{1}),$$

where the equality follows from [Mar75]. We are interested not only in $\mathsf{V}(\mathsf{s}_0)$, but also its ε-approximations and the corresponding (ε-)optimal strategies for both players.

Now we recall a fundamental tool for analysis of MDP called end components. We introduce the following notation. Given a set of states $T \subseteq S$, a state $\mathsf{s} \in T$ and an action $\mathsf{a} \in \mathsf{Av}(\mathsf{s})$, we say that (s, a) exits T if $\mathsf{Post}(\mathsf{s}, \mathsf{a}) \not\subseteq T$. We define an end component of a SG as the end component of the underlying MDP with both players unified.

Definition 2 (EC). *A non-empty set $T \subseteq S$ of states is an* end component (EC) *if there is a non-empty set $B \subseteq \bigcup_{\mathsf{s} \in T} \mathsf{Av}(s)$ of actions such that*

1. *for each $\mathsf{s} \in T, \mathsf{a} \in B \cap \mathsf{Av}(\mathsf{s})$ we do* not *have (s, a) exits T,*
2. *for each $\mathsf{s}, \mathsf{s}' \in T$ there is a finite path $\mathsf{w} = \mathsf{s}\mathsf{a}_0 \ldots \mathsf{a}_n\mathsf{s}' \in (T \times B)^* \times T$, i.e. the path stays inside T and only uses actions in B.*

Intuitively, ECs correspond to bottom strongly connected components of the Markov chains induced by possible strategies, so for some pair of strategies all possible paths starting in the EC remain there. An end component T is a *maximal end component (MEC)* if there is no other end component T' such that $T \subseteq T'$. Given an SG G, the set of its MECs is denoted by $\mathsf{MEC}(\mathsf{G})$ and can be computed in polynomial time [CY95].

2.2 (Bounded) value iteration

The value function V satisfies the following system of equations, which is referred to as the *Bellman equations*:

$$\mathsf{V}(\mathsf{s}) = \begin{cases} \max_{\mathsf{a} \in \mathsf{Av}(\mathsf{s})} \mathsf{V}(\mathsf{s}, \mathsf{a}) & \text{if } \mathsf{s} \in S_\square \\ \min_{\mathsf{a} \in \mathsf{Av}(\mathsf{s})} \mathsf{V}(\mathsf{s}, \mathsf{a}) & \text{if } \mathsf{s} \in S_\bigcirc \\ 1 & \text{if } \mathsf{s} = \mathfrak{1} \\ 0 & \text{if } \mathsf{s} = \mathfrak{0} \end{cases} \tag{1}$$

where[2]

$$\mathsf{V}(\mathsf{s}, \mathsf{a}) := \sum_{s' \in S} \delta(\mathsf{s}, \mathsf{a}, \mathsf{s}') \cdot \mathsf{V}(\mathsf{s}') \tag{2}$$

Moreover, V is the *least* solution to the Bellman equations, see e.g. [CH08]. To compute the value of V for all states in an SG, one can thus utilize the iterative approximation method *value iteration (VI)* as follows. We start with a lower bound function $\mathsf{L}_0 \colon S \to [0, 1]$ such that $\mathsf{L}_0(\mathfrak{1}) = 1$ and, for all other $\mathsf{s} \in S$, $\mathsf{L}_0(\mathsf{s}) = 0$. Then we repetitively apply Bellman updates (3) and (4)

$$\mathsf{L}_n(\mathsf{s}, \mathsf{a}) := \sum_{s' \in S} \delta(\mathsf{s}, \mathsf{a}, \mathsf{s}') \cdot \mathsf{L}_{n-1}(\mathsf{s}') \tag{3}$$

$$\mathsf{L}_n(\mathsf{s}) := \begin{cases} \max_{\mathsf{a} \in \mathsf{Av}(\mathsf{s})} \mathsf{L}_n(\mathsf{s}, \mathsf{a}) & \text{if } \mathsf{s} \in S_\square \\ \min_{\mathsf{a} \in \mathsf{Av}(\mathsf{s})} \mathsf{L}_n(\mathsf{s}, \mathsf{a}) & \text{if } \mathsf{s} \in S_\bigcirc \end{cases} \tag{4}$$

[2] Throughout the paper, for any function $f : S \to [0, 1]$ we overload the notation and also write $f(\mathsf{s}, \mathsf{a})$ meaning $\sum_{s' \in S} \delta(\mathsf{s}, \mathsf{a}, \mathsf{s}') \cdot f(\mathsf{s}')$.

until convergence. Note that convergence may happen only in the limit even for such a simple game as in Figure 1 on the left. The sequence is monotonic, at all times a *lower* bound on V, i.e. $\mathsf{L}_i(\mathsf{s}) \leq \mathsf{V}(\mathsf{s})$ for all $\mathsf{s} \in S$, and the least fixpoint satisfies $\mathsf{L}^* := \lim_{n \to \infty} \mathsf{L}_n = \mathsf{V}$.

Unfortunately, there is no known stopping criterion, i.e. no guarantees how close the current under-approximation is to the value [HM17]. The current tools stop when the difference between two successive approximations is smaller than a certain threshold, which can lead to arbitrarily wrong results [HM17].

For the special case of MDP, it has been suggested to also compute the greatest fixpoint [MLG05] and thus an *upper* bound as follows. The function $\mathsf{G} : S \to [0,1]$ is initialized for all states $\mathsf{s} \in S$ as $\mathsf{G}_0(\mathsf{s}) = 1$ except for $\mathsf{G}_0(\mathfrak{o}) = 0$. Then we repetitively apply updates (3) and (4), where L is replaced by G. The resulting sequence G_n is monotonic, provides an upper bound on V and the greatest fixpoint $\mathsf{G}^* := \lim_n \mathsf{G}_n$ is the greatest solution to the Bellman equations on $[0,1]^S$.

This approach is called *bounded value iteration (BVI)* (or *bounded real-time dynamic programming (BRTDP)* [MLG05,BCC$^+$14] or *interval iteration* [HM17]). If $\mathsf{L}^* = \mathsf{G}^*$ then they are both equal to V and we say that *BVI converges*. BVI is guaranteed to converge in MDP if the only ECs are those of $\mathfrak{1}$ and $\mathfrak{o}$ [BCC$^+$14]. Otherwise, if there are non-trivial ECs they have to be "collapsed"[3]. Computing the greatest fixpoint on the modified MDP results in another sequence U_i of upper bounds on V, converging to $\mathsf{U}^* := \lim_n \mathsf{U}_n$. Then BVI converges even for general MDPs, $\mathsf{U}^* = \mathsf{V}$ [BCC$^+$14], when transformed this way. The next section illustrates this difficulty and the solution through collapsing on an example.

In summary, all versions of BVI discussed so far and later on in the paper follow the pattern of Algorithm 1. In the naive version, UPDATE just performs the Bellman update on L and U according to Equations (3) and (4).[4] For a general MDP, U does not converge to V, but to G^*, and thus the termination criterion may never be met if $\mathsf{G}^*(\mathsf{s}_0) - \mathsf{V}(\mathsf{s}_0) > 0$. If the ECs are collapsed in pre-processing then U converges to V.

For the general case of SG, the collapsing approach fails and this paper provides another version of BVI where U converges to V, based on a more detailed structural analysis of the game.

3 Example

In this section, we illustrate the issues preventing BVI convergence and our solution on a few examples. Recall that G is the sequence converging to the greatest solution of the Bellman equations, while U is in general any sequence over-approximating V that one or another BVI algorithm suggests.

Firstly, we illustrate the issue that arises already for the special case of MDP. Consider the MPD of Figure 1 on the left. Although $\mathsf{V}(\mathsf{s}) = \mathsf{V}(\mathsf{t}) = 0.5$, we have

[3] All states of an EC are merged into one, all leaving actions are preserved and all other actions are discarded. For more detail see [KKKW18, Appendix A.1.]

[4] For the straightforward pseudocode, see [KKKW18, Appendix A.2.].

Algorithm 1 Bounded value iteration algorithm

```
1: procedure BVI(precision $\epsilon > 0$)
2:     for $\mathsf{s} \in S$ do      \* Initialization * \
3:         $\mathsf{L}(\mathsf{s}) = 0$        \* Lower bound * \
4:         $\mathsf{U}(\mathsf{s}) = 1$        \* Upper bound * \
5:     $\mathsf{L}(\mathtt{1}) = 1$              \* Value of sinks is determined a priori * \
6:     $\mathsf{U}(\mathtt{0}) = 0$
7:     repeat
8:         $\mathsf{UPDATE}(\mathsf{L}, \mathsf{U})$            \* Bellman updates or their modification * \
9:     until $\mathsf{U}(\mathsf{s}_0) - \mathsf{L}(\mathsf{s}_0) < \epsilon$       \* Guaranteed error bound * \
```

$\mathsf{G}_i(\mathsf{s}) = \mathsf{G}_i(\mathsf{t}) = 1$ for all i. Indeed, the upper bound for t is always updated as the maximum of $\mathsf{G}_i(\mathsf{t},\mathsf{c})$ and $\mathsf{G}_i(\mathsf{t},\mathsf{b})$. Although $\mathsf{G}_i(\mathsf{t},\mathsf{c})$ decreases over time, $\mathsf{G}_i(\mathsf{t},\mathsf{b})$ remains the same, namely equal to $\mathsf{G}_i(\mathsf{s})$, which in turn remains equal to $\mathsf{G}_i(\mathsf{s},\mathsf{a}) = \mathsf{G}_i(\mathsf{t})$. This cyclic dependency lets both s and t remain in an "illusion" that the value of the other one is 1.

The solution for MDP is to remove this cyclic dependency by collapsing all MECs into singletons and removing the resulting purely self-looping actions. Figure 1 in the middle shows the MDP after collapsing the EC $\{\mathsf{s},\mathsf{t}\}$. This turns the MDP into a stopping one, where $\mathtt{1}$ or $\mathtt{0}$ is under any strategy reached with probability 1. In such MDP, there is a unique solution to the Bellman equations. Therefore, the greatest fixpoint is equal to the least one and thus to V.

Secondly, we illustrate the issues that additionally arise for general SG. It turns out that the collapsing approach can be extended only to games where all states of each EC belong to one player only [Ujm15]. In this case, both Maximizer's and Minimizer's ECs are collapsed the same way as in MDP.

However, when both players are present in an EC, then collapsing may not solve the issue. Consider the SG of Figure 2. Here α and β represent the values of the respective actions.[5] There are three cases:

First, let $\alpha < \beta$. If the bounds converge to these values we eventually observe $\mathsf{G}_i(q,e) < \mathsf{L}_i(r,f)$ and learn the induced inequality. Since p is a Minimizer's state it will never pick the action leading to the greater value of β. Therefore, we can safely merge p and q, and remove the action leading to r, as shown in the second subfigure.

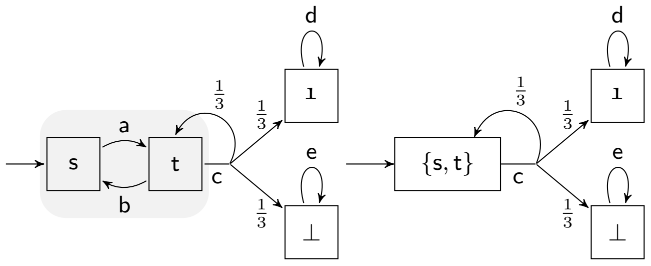
Second, if $\alpha > \beta$, p and r can be merged in an analogous way, as shown in the third subfigure.

Third, if $\alpha = \beta$, both previous solutions as well as collapsing all three states as in the fourth subfigure is possible. However, since the approximants may only converge to α and β in the limit, we may not know in finite time which of these cases applies and thus cannot decide for any of the collapses.

Consequently, the approach of collapsing is not applicable in general. In order to ensure BVI convergence, we suggest a different method, which we call

[5] Precisely, we consider them to stand for a probabilistic branching with probability α (or β) to $\mathtt{1}$ and with the remaining probability to $\mathtt{0}$. To avoid clutter in the figure, we omit this branching and depict only the value.

deflating. It does not involve changing the state space, but rather decreasing the upper bound U_i to the least value that is currently provable (and thus still correct). To this end, we analyze the exiting actions, i.e. with successors outside of the EC, for the following reason. If the play stays in the EC forever, the target is never reached and Minimizer wins. Therefore, Maximizer needs to pick some exiting action to avoid staying in the EC.



i	$\mathsf{L}_i(\{\mathsf{s},\mathsf{t}\})$	$\mathsf{G}_i(\{\mathsf{s},\mathsf{t}\})$
0	0	1
1	$\frac{1}{3}$	$\frac{2}{3}$
2	$\frac{4}{9}$	$\frac{5}{9}$
3	$\frac{13}{27}$	$\frac{14}{27}$

Fig. 1: Left: An MDP (as special case of SG) where BVI does not converge due to the grayed EC. Middle: The same MDP where the EC is collapsed, making BVI converge. Right: The approximations illustrating the convergence of the MDP in the middle.

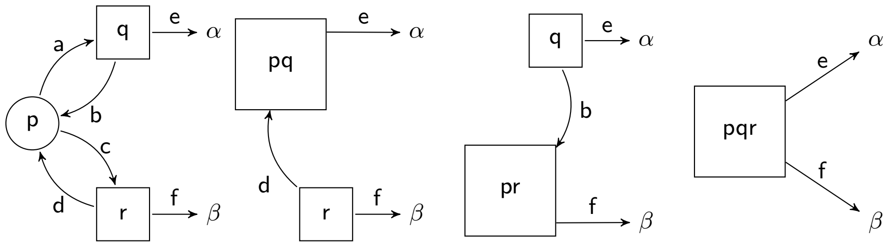


Fig. 2: Left: Collapsing ECs in SG may lead to incorrect results. The Greek letters on the leaving arrows denote the values of the exiting actions. Right three figures: Correct collapsing in different cases, depending on the relationship of α and β. In contrast to MDP, some actions of the EC exiting the collapsed part have to be removed.

For the EC with the states s and t in Figure 1, the only exiting action is c. In this example, since c is the only exiting action, $\mathsf{U}_i(\mathsf{t},\mathsf{c})$ is the highest possible upper bound that the EC can achieve. Thus, by decreasing the upper bound of all states in the EC to that number[6], we still have a safe upper bound. Moreover, with this modification BVI converges in this example, intuitively because now the upper bound of t depends on action c as it should.

For the example in Figure 2, it is correct to decrease the upper bound to the maximal exiting one, i.e. $\max\{\hat{\alpha},\hat{\beta}\}$, where $\hat{\alpha} := \mathsf{U}_i(\mathsf{a}), \hat{\beta} := \mathsf{U}_i(\mathsf{b})$ are the

[6] We choose the name "deflating" to evoke decreasing the overly high "pressure" in the EC until it equalizes with the actual "pressure" outside.

current approximations of α and of β. However, this itself does not ensure BVI convergence. Indeed, if for instance $\hat{\alpha} < \hat{\beta}$ then deflating all states to $\hat{\beta}$ is not tight enough, as values of p and q can even be bounded by $\hat{\alpha}$. In fact, we have to find a certain sub-EC that corresponds to $\hat{\alpha}$, in this case $\{\mathsf{p}, \mathsf{q}\}$ and set all its upper bounds to $\hat{\alpha}$. We define and compute these sub-ECs in the next section.

In summary, the general structure of our convergent BVI algorithm is to produce the sequence U by application of Bellman updates and occasionally find the relevant sub-ECs and deflate them. The main technical challenge is that states in an EC in SG can have different values, in contrast to the case of MDP.

4 Convergent Over-approximation

In Section 4.1, we characterize SGs where Bellman equations have more solutions. Based on the analysis, subsequent sections show how to alter the procedure computing the sequence G_i over-approximating V so that the resulting tighter sequence U_i still over-approximates V, but also converges to V. This ensures that thus modified BVI converges. Section 4.4 presents the learning-based variant of our BVI.

4.1 Bloated end components cause non-convergence

As we have seen in the example of Fig. 2, BVI generally does not converge due to ECs with a particular structure of the exiting actions. The analysis of ECs relies on the extremal values that can be achieved by exiting actions (in the example, α and β). Given the value function V or just its current over-approximation U_i, we define the most profitable exiting action for Maximizer (denoted by $\square$) and Minimizer (denoted by $\bigcirc$) as follows.

Definition 3 ($\mathsf{bestExit}$)**.** *Given a set of states $T \subseteq S$ and a function $f : S \to [0,1]$ (see footnote 2), the f-value of the best T-exiting action of Maximizer and Minimizer, respectively, is defined as*

$$\mathsf{bestExit}_f^{\square}(T) = \max_{\substack{\mathsf{s} \in T_{\square} \\ (\mathsf{s},\mathsf{a}) \text{ exits } T}} f(\mathsf{s},\mathsf{a})$$

$$\mathsf{bestExit}_f^{\bigcirc}(T) = \min_{\substack{\mathsf{s} \in T_{\bigcirc} \\ (\mathsf{s},\mathsf{a}) \text{ exits } T}} f(\mathsf{s},\mathsf{a})$$

with the convention that $\max_\emptyset = 0$ *and* $\min_\emptyset = 1$*.*

Example 1. In the example of Fig. 2 on the left with $T = \{\mathsf{p}, \mathsf{q}, \mathsf{r}\}$ and $\alpha < \beta$, we have $\mathsf{bestExit}_{\mathsf{V}}^{\square}(T) = \beta$, $\mathsf{bestExit}_{\mathsf{V}}^{\bigcirc}(T) = 1$. It is due to $\beta < 1$ that BVI does not converge here. We generalize this in the following lemma. $\triangle$

Lemma 1. *Let T be an EC. For every m satisfying* $\mathsf{bestExit}_{\mathsf{V}}^{\square}(T) \leq m \leq \mathsf{bestExit}_{\mathsf{V}}^{\bigcirc}(T)$*, there is a solution $f \colon S \to [0,1]$ to the Bellman equations, which on T is constant and equal to m.*

Proof (Idea). Intuitively, such a constant m is a solution to the Bellman equations on T for the following reasons. As both players prefer getting m to exiting and getting "only" the values of their respective $\mathsf{bestExit}$, they both choose to stay in the EC (and the extrema in the Bellman equations are realized on non-exiting actions). On the one hand, Maximizer (Bellman equations with max) is hoping for the promised m, which is however not backed up by any actions actually exiting towards the target. On the other hand, Minimizer (Bellman equations with min) does not realize that staying forever results in her optimal value 0 instead of m. □

Corollary 1. *If* $\mathsf{bestExit}^{\bigcirc}_{\mathsf{V}}(T) > \mathsf{bestExit}^{\square}_{\mathsf{V}}(T)$ *for some EC* T*, then* $\mathsf{G}^* \neq \mathsf{V}$*.*

Proof. Since there are m_1, m_2 such that $\mathsf{bestExit}^{\square}_{\mathsf{V}}(T) < m_1 < m_2 < \mathsf{bestExit}^{\bigcirc}_{\mathsf{V}}(T)$, by Lemma 1 there are two different solutions to the Bellman equations. In particular, $\mathsf{G}^* > \mathsf{L}^* = \mathsf{V}$, and BVI does not converge. □

In accordance with our intuition that ECs satisfying the above inequality should be deflated, we call them bloated.

Definition 4 (BEC). *An EC* T *is called a* bloated end component (BEC)*, if* $\mathsf{bestExit}^{\bigcirc}_{\mathsf{V}}(T) > \mathsf{bestExit}^{\square}_{\mathsf{V}}(T)$*.*

Example 2. In the example of Fig. 2 on the left with $\alpha < \beta$, the ECs $\{\mathsf{p}, \mathsf{q}\}$ and $\{\mathsf{p}, \mathsf{q}, \mathsf{r}\}$ are BECs. △

Example 3. If an EC T has no exiting actions of Minimizer (or no Minimizer's states at all, as in an MDP), then $\mathsf{bestExit}^{\bigcirc}_{\mathsf{V}}(T) = 1$ (the case with $\min_{\emptyset}$). Hence all numbers between $\mathsf{bestExit}^{\square}_{\mathsf{V}}(T)$ and 1 are a solution to the Bellman equations and $\mathsf{G}^*(s) = 1$ for all states $s \in T$.

Analogously, if Maximizer does not have any exiting action in T , then it holds that $\mathsf{bestExit}^{\square}_{\mathsf{V}}(T) = 0$ (the case with $\max_{\emptyset}$), T is a BEC and all numbers between 0 and $\mathsf{bestExit}^{\bigcirc}_{\mathsf{V}}(T)$ are a solution to the Bellman equations.

Note that in MDP all ECs belong to one player, namely Maximizer. Consequently, all ECs are BECs except for ECs where Maximizer has an exiting action with value 1; all other ECs thus have to be collapsed (or deflated) to ensure BVI convergence in MDPs. Interestingly, all non-trivial ECs in MDPs are a problem, while in SGs through the presence of the other player some ECs can converge, namely if both players want to exit (See e.g. [KKKW18, Appendix A.3.]). △

We show that BECs are indeed the only obstacle for BVI convergence.

Theorem 1. *If the SG contains no BECs except for* $\{\mathfrak{0}\}$ *and* $\{\mathfrak{1}\}$*, then* $\mathsf{G}^* = \mathsf{V}$*.*

Proof (Sketch). Assume, towards a contradiction, that there is some state s with a positive difference $\mathsf{G}^*(\mathsf{s}) - \mathsf{V}(\mathsf{s}) > 0$. Consider the set D of states with the maximal difference. D can be shown to be an EC. Since it is not a BEC there has to be an action exiting D and realizing the optimum in that state. Consequently, this action also has the maximal difference, and all its successors, too. Since some of the successors are outside of D, we get a contradiction with the maximality of D. □

In Section 4.2, we show how to eliminate BECs by collapsing their "core" parts, called below MSECs (maximal simple end components). Since MSECs can only be identified with enough information about V, Section 4.3 shows how to avoid direct *a priori* collapsing and instead dynamically deflate candidates for MSECs in a conservative way.

4.2 Static MSEC decomposition

Now we turn our attention to SG with BECs. Intuitively, since in a BEC all Minimizer's exiting actions have a higher value than what Maximizer can achieve, Minimizer does not want to use any of his own exiting actions and prefers staying in the EC (or steering Maximizer towards his worse exiting actions). Consequently, only Maximizer wants to take an exiting action. In the MDP case he can pick any desirable one. Indeed, he can wait until he reaches a state where it is available. As a result, in MDP all states of an EC have the *same value* and can all be collapsed into one state. In the SG case, he may be restricted by Minimizer's behaviour or even not given any chance to exit the EC at all. As a result, a BEC may contain several parts (below denoted MSECs), each with different value, intuitively corresponding to different exits. Thus instead of MECs, we have to decompose into finer MSECs and only collapse these.

Definition 5 (*Simple EC*)**.** *An EC T is called* simple (SEC)*, if for all $\mathsf{s} \in T$ we have $\mathsf{V}(\mathsf{s}) = \mathsf{bestExit}_{\mathsf{V}}^{\Box}(T)$.*
A SEC C is maximal (MSEC) *if there is no SEC C' such that $C \subsetneq C'$.*

Intuitively, an EC is simple, if Minimizer cannot keep Maximizer away from his $\mathsf{bestExit}$. Independently of Minimizer's decisions, Maximizer can reach the $\mathsf{bestExit}$ almost surely, unless Minimizer decides to leave, in which case Maximizer could achieve an even higher value.

Example 4. Assume $\alpha < \beta$ in the example of Figure 2. Then $\{\mathsf{p}, \mathsf{q}\}$ is a SEC and an MSEC. Further observe that action c is sub-optimal for Minimizer and removing it does not affect the value of any state, but simplifies the graph structure. Namely, it destructs the whole EC into several (here only one) SECs and some non-EC states (here r). △

Algorithm 2, called $\mathsf{FIND\_MSEC}$, shows how to compute MSECs. It returns the set of all MSECs if called with parameter V. However, later we also call this function with other parameters $f : S \to [0, 1]$. The idea of the algorithm is the following. The set X consists of Minimizer's sub-optimal actions, leading to a higher value. As such they cannot be a part of any SEC and thus should be ignored when identifying SECs. (The previous example illustrates that ignoring X is indeed safe as it does not change the value of the game.) We denote the game G where the available actions Av are changed to the new available actions Av' (ignoring the Minimizer's sub-optimal ones) as $\mathsf{G}_{[\mathsf{Av}/\mathsf{Av}']}$. Once removed, Minimizer has no choices to affect the value and thus each EC is simple.

Algorithm 2 FIND_MSEC

```
1: function FIND_MSEC($f : S \to [0,1]$)
2:     $X \leftarrow \{(\mathsf{s}, \{\mathsf{a} \in \mathsf{Av}(\mathsf{s}) \mid f(\mathsf{s},\mathsf{a}) > f(\mathsf{s})\}) \mid \mathsf{s} \in S_{\bigcirc}\}$
3:     $\mathsf{Av}' \leftarrow \mathsf{Av} \setminus X$                \* Minimizer's $f$-suboptimal actions removed * \
4:     return $\mathsf{MEC}(\mathsf{G}_{[\mathsf{Av}/\mathsf{Av}']})$        \* $\mathsf{MEC}(\mathsf{G}_{[\mathsf{Av}/\mathsf{Av}']})$ are MSECs of the original $\mathsf{G}$ * \
```

Lemma 2 (Correctness of Algorithm 2). *$T \in \mathsf{FIND\_MSEC}(\mathsf{V})$ if and only if T is a MSEC.*

Proof (Sketch). "If": As T is an MSEC, all states in T have the value $\mathsf{bestExit}^{\square}_{\mathsf{V}}(T)$, and hence also all actions that stay inside T have this value. Thus, no action that stays in T is removed by Line 3 and it is still a MEC in the modified game.

"Only if": If $T \in \mathsf{FIND\_MSEC}(\mathsf{V})$, then T is a MEC of the game where the suboptimal available actions (those in X) of Minimizer have been removed. Hence for all $\mathsf{s} \in T : \mathsf{V}(\mathsf{s}) = \mathsf{bestExit}^{\square}_{\mathsf{V}}(T)$, because intuitively Minimizer has no possibility to influence the value any further, since all actions that could do so were in X and have been removed. Since T is a MEC in the modified game, it certainly is an EC in the original game. Hence T is a SEC. The inclusion maximality follows from the fact that we compute MECs in the modified game. Thus T is an MSEC. $\square$

Remark 1 (Algorithm with an oracle). In Section 3, we have seen that collapsing MECs does not ensure BVI convergence. Collapsing does not preserve the values, since in BECs we would be collapsing states with different values. Hence we want to collapse only MSECs, where the values are the same. If, moreover, we remove X in such a collapsed SG, then there are no (non-sink) ECs and BVI converges on this SG to the original value.

The difficulty with this algorithm is that it requires an oracle to compare values, for instance a sufficiently precise approximation of V. Consequently, we cannot pre-compute the MSECs, but have to find them while running BVI. Moreover, since the approximations converge only in the limit we may never be able to conclude on simplicity of some ECs. For instance, if $\alpha = \beta$ in Figure 2, and if the approximations converge at different speeds, then Algorithm 2 always outputs only a part of the EC, although the whole EC on $\{\mathsf{p},\mathsf{q},\mathsf{r}\}$ is simple.

In MDPs, all ECs are simple, because there is no second player to be resolved and all states in an EC have the same value. Thus for MDPs it suffices to collapse all MECs, in contrast to SG.

4.3 Dynamic MSEC decomposition

Since MSECs cannot be identified from approximants of V for sure, we refrain from collapsing[7] and instead only decrease the over-approximation in the corresponding way. We call the method *deflating*, by which we mean decreasing the upper bound of all states in an EC to its $\mathsf{bestExit}^{\square}_{\mathsf{U}}$, see Algorithm 3.

[7] Our subsequent method can be combined with local collapsing whenever the lower and upper bounds on V are conclusive.

The procedure DEFLATE (called on the current upper bound U_i) decreases this upper bound to the minimum possible value according to the current approximation and thus prevents states from only depending on each other, as in SECs. Intuitively, it gradually approximates SECs and performs the corresponding adjustments, but does not commit to any of the approximations.

Algorithm 3 DEFLATE

```
1: function DEFLATE(EC $T$, $f : S \to [0,1]$)
2:     for $\mathsf{s} \in T$ do
3:         $f(\mathsf{s}) \leftarrow \min(f(\mathsf{s}), \mathsf{bestExit}_f^{\Box}(T))$      \* Decrease the upper bound * \
4:     return $f$
```

Lemma 3 **(DEFLATE is sound).** *For any $f : S \to [0,1]$ such that $f \geq \mathsf{V}$ and any EC T, $\mathsf{DEFLATE}(T, f) \geq \mathsf{V}$.*

This allows us to define our BVI algorithm as the naive BVI with only the additional lines 3-4, see Algorithm 4.

Algorithm 4 UPDATE procedure for bounded value iteration on SG

```
1: procedure UPDATE($\mathsf{L} : S \to [0,1]$, $\mathsf{U} : S \to [0,1]$)
2:     $\mathsf{L}, \mathsf{U}$ get updated according to Eq. (3) and (4)      \* Bellman updates * \
3:     for $T \in$ FIND_MSEC($\mathsf{L}$) do              \* Use lower bound to find ECs * \
4:         $\mathsf{U} \leftarrow$ DEFLATE($T, \mathsf{U}$)                  \* and deflate the upper bound there * \
```

Theorem 2 **(Soundness and completeness).** *Algorithm 1 (calling Algorithm 4) produces monotonic sequences L under- and U over-approximating V, and terminates.*

Proof (Sketch). The crux is to show that U converges to V. We assume towards a contradiction, that there exists a state s with $\lim_{n\to\infty} \mathsf{U}_n(\mathsf{s}) - \mathsf{V}(\mathsf{s}) > 0$. Then there exists a nonempty set of states X where the difference between $\lim_{n\to\infty} \mathsf{U}_n$ and V is maximal. If the upper bound of states in X depends on states outside of X, this yields a contradiction, because then the difference between upper bound and value would decrease in the next Bellman update. So X must be an EC where all states depend on each other. However, if that is the case, calling DEFLATE decreases the upper bound to something depending on the states outside of X, thus also yielding a contradiction. $\square$

Summary of our approach:

1. We cannot collapse MECs, because we cannot collapse BECs with non-constant values.
2. If we remove X (the sub-optimal actions of Minimizer) we can collapse MECs (now actually MSECs with constant values).
3. Since we know neither X nor SECs we gradually deflate SEC approximations.

4.4 Learning-based algorithm

Asynchronous value iteration selects in each round a subset $T \subseteq S$ of states and performs the Bellman update in that round only on T. Consequently, it may speed up computation if "important" states are selected. However, using the standard VI it is even more difficult to determine the current error bound. Moreover, if some states are not selected infinitely often the lower bound may not even converge.

In the setting of bounded value iteration, the current error bound is known for each state and thus convergence can easily be enforced. This gave rise to asynchronous VI, such as BRTDP (bounded real time dynamic programing) in the setting of stopping MDPs [MLG05], where the states are selected as those that appear on a simulation run. Very similar is the adaptation for general MDP [BCC$^+$14]. In order to simulate a run, the transition probabilities determine how to resolve the probabilistic choice. In order to resolve the non-deterministic choice of Maximizer, the "most promising action" is taken, i.e., with the highest U. This choice is derived from a reinforcement algorithm called delayed Q-learning and ensures convergence while practically performing well [BCC$^+$14].

In this section, we harvest our convergence results and BVI algorithm for SG, which allow us to trivially extend the asynchronous learning-based approach of BRTDP to SGs. On the one hand, the only difference to the MDP algorithm is how to resolve the choice for Minimizer. Since the situation is dual, we again pick the "most promising action", in this case with the lowest L. On the other hand, the only difference to Algorithm 1 calling Algorithm 4 is that the Bellman updates of U and L are performed on the states of the simulation run only, see lines 2-3 of Algorithm 5.

Algorithm 5 Update procedure for the learning/BRTDP version of BVI on SG

```
1: procedure UPDATE($\mathsf{L} : S \to [0,1]$, $\mathsf{U} : S \to [0,1]$)
2:     $\rho \leftarrow$ path $\mathsf{s}_0, \mathsf{s}_1, \ldots, \mathsf{s}_\ell$ of length $\ell \leq k$, obtained by simulation where the
       successor of $\mathsf{s}$ is $\mathsf{s}'$ with probability $\delta(\mathsf{s}, \mathsf{a}, \mathsf{s}')$ and $\mathsf{a}$ is sampled randomly from
       $\arg\max_{\mathsf{a}} \mathsf{U}(\mathsf{s}, \mathsf{a})$ and $\arg\min_{\mathsf{a}} \mathsf{L}(\mathsf{s}, \mathsf{a})$ for $\mathsf{s} \in S_\Box$ and $\mathsf{s} \in S_\bigcirc$, respectively
3:     $\mathsf{L}, \mathsf{U}$ get updated by Eq. (3) and (4) on states $\mathsf{s}_\ell, \mathsf{s}_{\ell-1}, \ldots, \mathsf{s}_0$    \* all $\mathsf{s} \in \rho$ * \
4:     for $T \in$ FIND_MSEC($\mathsf{L}$) do
5:         DEFLATE($T, \mathsf{U}$)
```

If $\mathtt{1}$ or $\mathtt{0}$ is reached in a simulation, we can terminate it. It can happen that the simulation cycles in an EC. To that end, we have a bound k on the maximum number of steps. The choice of k is discussed in detail in [BCC$^+$14] and we use $2 \cdot |S|$ to guarantee the possibility of reaching sinks as well as exploring new states. If the simulation cycles in an EC, the subsequent call of DEFLATE ensures that next time there is a positive probability to exit this EC. Further details can be found in [KKKW18, Appendix A.4.].

5 Experimental results

We implemented both our algorithms as an extension of PRISM-games [CFK$^+$13a], a branch of PRISM [KNP11] that allows for modelling SGs, utilizing previous work of [BCC$^+$14,Ujm15] for MDP and SG with single-player ECs. We tested the implementation on the SGs from the PRISM-games case studies [gam] that have reachability properties and one additional model from [CKJ12] that was also used in [Ujm15]. We compared the results with both the explicit and the hybrid engine of PRISM-games, but since the models are small both of them performed similar and we only display the results of the hybrid engine in Table 1.

Furthermore we ran experiments on MDPs from the PRISM benchmark suite [KNP12]. We compared our results there to the hybrid and explicit engine of PRISM, the interval iteration implemented in PRISM [HM17], the hybrid engine of STORM [DJKV17b] and the BRTDP implementation of [BCC$^+$14].

Recall that the aim of the paper is not to provide a faster VI algorithm, but rather the first guaranteed one. Consequently, the aim of the experiments is not to show any speed ups, but to experimentally estimate the overhead needed for computing the guarantees.

For information on the technical details of the experiments, all the models and the tables for the experiments on MDPs we refer to [KKKW18, Appendix B]. Note that although some of the SG models are parametrized they could only be scaled by manually changing the model file, which complicates extensive benchmarking.

Although our approaches compute the additional upper bound to give the convergence guarantees, for each of the experiments one of our algorithms performed similar to PRISM-games. Table 1 shows this result for three of the four SG models in the benchmarking set. On the fourth model, PRISM's precomputations already solve the problem and hence it cannot be used to compare the approaches. For completeness, the results are displayed in [KKKW18, Appendix B.5].

Table 1: Experimental results for the experiments on SGs. The left two columns denote the model and the given parameters, if present. Columns 3 to 5 display the verification time in seconds for each of the solvers, namely PRISM-games (referred as PRISM), our BVI algorithm (BVI) and our learning-based algorithm (BRTDP). The next two columns compare the number of states that BRTDP explored (#States_B) to the total number of states in the model. The rightmost column shows the number of MSECs in the model.

Model	Parameters	PRISM	BVI	BRTDP	#States_B	#States	#MSECs
mdsm	prop=1	8	8	17	767	62,245	1
	prop=2	4	4	29	407	62,245	1
cdmsn		2	2	3	1,212	1,240	1
cloud	N=5	3	7	15	1,302	8,842	4,421
	N=6	6	59	4	570	34,954	17,477

Whenever there are few MSECs, as in mdsm and cdmsn, BVI performs like PRISM-games, because only little time is used for deflating. Apparently the additional upper bound computation takes very little time in comparison to the other tasks (e.g. parsing, generating the model, pre-computation) and does not slow down the verification significantly. For cloud, BVI is slower than PRISM-games, because there are thousands of MSECs and deflating them takes over 80% of the time. This comes from the fact that we need to compute the expensive end component decomposition for each deflating step. BRTDP performs well for cloud, because in this model, as well as generally often if there are many MECs [BCC$^+$14], only a small part of the state space is relevant for convergence. For the other models, BRTDP is slower than the deterministic approaches, because the models are so small that it is faster to first construct them completely than to explore them by simulation.

Our more extensive experiments on MDPs compare the guaranteed approaches based on collapsing (i.e. learning-based from [BCC$^+$14] and deterministic from [HM17]) to our guaranteed approaches based on deflating (so BRTDP and BVI). Since both learning-based approaches as well as both deterministic approaches perform similarly (see Table 2 in [KKKW18, Appendix B]), we conclude that collapsing and deflating are both useful for practical purposes, while the latter is also applicable to SGs. Furthermore we compared the usual unguaranteed value iteration of PRISM's explicit engine to BVI and saw that our guaranteed approach did not take significantly more time in most cases. This strengthens the point that the overhead for the computation of the guarantees is negligible

6 Conclusions

We have provided the first stopping criterion for value iteration on simple stochastic games and an anytime algorithm with bounds on the current error (guarantees on the precision of the result). The main technical challenge was that states in end components in SG can have different values, in contrast to the case of MDP. We have shown that collapsing is in general not possible, but we utilized the analysis to obtain the procedure of *deflating*, a solution on the original graph. Besides, whenever a SEC is identified for sure it can be collapsed and the two techniques of collapsing and deflating can thus be combined.

The experiments indicate that the price to pay for the overhead to compute the error bound is often negligible. For each of the available models, at least one of our two implementations has performed similar to or better than the standard approach that yields no guarantees. Further, the obtained guarantees open the door to (e.g. learning-based) heuristics which treat only a part of the state space and can thus potentially lead to huge improvements. Surprisingly, already our straightforward adaptation of such an algorithm for MDP to SG yields interesting results, palliating the overhead of our non-learning method, despite the most naive implementation of deflating. Future work could reveal whether other heuristics or more efficient implementation can lead to huge savings as in the case of MDP [BCC$^+$14].

References

[ACD$^+$17] Pranav Ashok, Krishnendu Chatterjee, Przemyslaw Daca, Jan Kretínský, and Tobias Meggendorfer. Value iteration for long-run average reward in markov decision processes. In *CAV*, pages 201–221, 2017.

[AM09] Daniel Andersson and Peter Bro Miltersen. The complexity of solving stochastic games on graphs. In *ISAAC*, pages 112–121, 2009.

[AY17] Gürdal Arslan and Serdar Yüksel. Decentralized q-learning for stochastic teams and games. *IEEE Trans. Automat. Contr.*, 62(4):1545–1558, 2017.

[BBS08] Lucian Busoniu, Robert Babuska, and Bart De Schutter. A comprehensive survey of multiagent reinforcement learning. *IEEE Trans. Systems, Man, and Cybernetics, Part C*, 38(2):156–172, 2008.

[BCC$^+$14] Tomás Brázdil, Krishnendu Chatterjee, Martin Chmelik, Vojtech Forejt, Jan Kretínský, Marta Z. Kwiatkowska, David Parker, and Mateusz Ujma. Verification of Markov decision processes using learning algorithms. In *ATVA*, pages 98–114. Springer, 2014.

[BK08] Christel Baier and Joost-Pieter Katoen. Principles of model checking, 2008.

[BKL$^+$17] Christel Baier, Joachim Klein, Linda Leuschner, David Parker, and Sascha Wunderlich. Ensuring the reliability of your model checker: Interval iteration for markov decision processes. In *CAV*, pages 160–180, 2017.

[BT00] Ronen I. Brafman and Moshe Tennenholtz. A near-optimal polynomial time algorithm for learning in certain classes of stochastic games. *Artif. Intell.*, 121(1-2):31–47, 2000.

[CF11] Krishnendu Chatterjee and Nathanaël Fijalkow. A reduction from parity games to simple stochastic games. In *GandALF*, pages 74–86, 2011.

[CFK$^+$13a] T. Chen, V. Forejt, M. Kwiatkowska, D. Parker, and A. Simaitis. PRISM-games: A model checker for stochastic multi-player games. In N. Piterman and S. Smolka, editors, *Proc. 19th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'13)*, volume 7795 of *LNCS*, pages 185–191. Springer, 2013.

[CFK$^+$13b] Taolue Chen, Vojtech Forejt, Marta Z. Kwiatkowska, David Parker, and Aistis Simaitis. Automatic verification of competitive stochastic systems. *Formal Methods in System Design*, 43(1):61–92, 2013.

[CH08] Krishnendu Chatterjee and Thomas A Henzinger. Value iteration. In *25 Years of Model Checking*, pages 107–138. Springer, 2008.

[CHJR10] Krishnendu Chatterjee, Thomas A. Henzinger, Barbara Jobstmann, and Arjun Radhakrishna. Gist: A solver for probabilistic games. In *CAV*, pages 665–669, 2010.

[CKJ12] Radu Calinescu, Shinji Kikuchi, and Kenneth Johnson. *Compositional Reverification of Probabilistic Safety Properties for Large-Scale Complex IT Systems*, pages 303–329. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[CKLB11] Chih-Hong Cheng, Alois Knoll, Michael Luttenberger, and Christian Buckl. GAVS+: an open platform for the research of algorithmic game solving. In *ETAPS*, pages 258–261, 2011.

[CKSW13] Taolue Chen, Marta Z. Kwiatkowska, Aistis Simaitis, and Clemens Wiltsche. Synthesis for multi-objective stochastic games: An application to autonomous urban driving. In *QEST*, pages 322–337, 2013.

[CMG14] Javier Cámara, Gabriel A. Moreno, and David Garlan. Stochastic game analysis and latency awareness for proactive self-adaptation. In *9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2014, Proceedings, Hyderabad, India, June 2-3, 2014*, pages 155–164, 2014.

[Con92] Anne Condon. The complexity of stochastic games. *Information and Computation*, 96(2):203–224, 1992.

[CY95] Costas Courcoubetis and Mihalis Yannakakis. The complexity of probabilistic verification. *Journal of the ACM*, 42(4):857–907, July 1995.

[DJKV17a] Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen, and Matthias Volk. A storm is coming: A modern probabilistic model checker. In *CAV*, pages 592–600, 2017.

[DJKV17b] Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen, and Matthias Volk. A storm is coming: A modern probabilistic model checker. *CoRR*, abs/1702.04311, 2017.

[gam] PRISM-games case studies. prismmodelchecker.org/games/casestudies.php. Accessed: 2017-09-18.

[HK66] A. J. Hoffman and R. M. Karp. On nonterminating stochastic games. *Management Science*, 12(5):359–370, 1966.

[HM17] Serge Haddad and Benjamin Monmege. Interval iteration algorithm for mdps and imdps. *Theoretical Computer Science*, 2017.

[KKKW18] Edon Kelmendi, Julia Krämer, Jan Křetínský, and Maximilian Weininger. Value iteration for simple stochastic games: Stopping criterion and learning algorithm. Technical Report abs/1804.04901, arXiv.org, 2018.

[KKNP10] Mark Kattenbelt, Marta Z. Kwiatkowska, Gethin Norman, and David Parker. A game-based abstraction-refinement framework for markov decision processes. *Formal Methods in System Design*, 36(3):246–280, 2010.

[KM17] Jan Kretínský and Tobias Meggendorfer. Efficient strategy iteration for mean payoff in markov decision processes. In *ATVA*, pages 380–399, 2017.

[KNP11] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *CAV*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

[KNP12] M. Kwiatkowska, G. Norman, and D. Parker. The prism benchmark suite. 9th International Conference on Quantitative Evaluation of Systems (QEST12),pages 203204. IEEE, 2012.

[LaV00] Steven M. LaValle. Robot motion planning: A game-theoretic foundation. *Algorithmica*, 26(3-4):430–465, 2000.

[LL08] Jianwei Li and Weiyi Liu. A novel heuristic q-learning algorithm for solving stochastic games. In *IJCNN*, pages 1135–1144, 2008.

[Mar75] Donald A Martin. Borel determinacy. *Annals of Mathematics*, pages 363–371, 1975.

[MLG05] H. Brendan Mcmahan, Maxim Likhachev, and Geoffrey J. Gordon. Bounded real-time dynamic programming: Rtdp with monotone upper bounds and performance guarantees. In *In ICML05*, pages 569–576, 2005.

[Put14] Martin L. Puterman. *Markov decision processes: Discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[SK16] María Svorenová and Marta Kwiatkowska. Quantitative verification and strategy synthesis for stochastic games. *Eur. J. Control*, 30:15–30, 2016.

[TT16] Alain Tcheukam and Hamidou Tembine. One swarm per queen: A particle swarm learning for stochastic games. In *SASO*, pages 144–145, 2016.

[Ujm15] Mateusz Ujma. *On Verication and Controller Synthesis for Probabilistic Systems at Runtime*. PhD thesis, Wolfson College, Oxford, 2015.

[WT16] Min Wen and Ufuk Topcu. Probably approximately correct learning in stochastic games with temporal logic specifications. In *IJCAI*, pages 3630–3636, 2016.

Rabinizer 4: From LTL to Your Favourite Deterministic Automaton[⋆]

Jan Křetínský, Tobias Meggendorfer, Salomon Sickert, and Christopher Ziegler

Technical University of Munich

Abstract. We present Rabinizer 4, a tool set for translating formulae of linear temporal logic to different types of deterministic ω-automata. The tool set implements and optimizes several recent constructions, including the first implementation translating the frequency extension of LTL. Further, we provide a distribution of PRISM that links Rabinizer and offers model checking procedures for probabilistic systems that are not in the official PRISM distribution. Finally, we evaluate the performance and in cases with any previous implementations we show enhancements both in terms of the size of the automata and the computational time, due to algorithmic as well as implementation improvements.

1 Introduction

Automata-theoretic approach [VW86] is a key technique for verification and synthesis of systems with linear-time specifications, such as formulae of linear temporal logic (LTL) [Pnu77]. It proceeds in two steps: first, the formula is translated into a corresponding automaton; second, the product of the system and the automaton is further analyzed. The size of the automaton is important as it directly affects the size of the product and thus largely also the analysis time, particularly for deterministic automata and probabilistic model checking in a very direct proportion. For verification of non-deterministic systems, mostly non-deterministic Büchi automata (NBA) are used [EH00,SB00,GO01,GL02,BKŘS12,DLLF$^+$16] since they are typically very small and easy to produce.

Probabilistic LTL model checking cannot profit directly from NBA. Even the qualitative question, whether a formula holds with probability 0 or 1, requires automata with at least a restricted form of determinism. The prime example are the limit-deterministic (also called semi-deterministic) Büchi automata (LDBA) [CY88] and the generalized LDBA (LDGBA). However, for the general quantitative questions, where the probability of satisfaction is computed, general limit-determinism is not sufficient. Instead, deterministic Rabin automata (DRA) have been mostly used [KNP11] and recently also deterministic generalized Rabin automata (DGRA) [CGK13]. In principle, all standard types of deterministic automata are applicable here except for deterministic Büchi automata (DBA), which are not as expressive as

⋆ This work has been partially supported by the Czech Science Foundation grant No. P202/12/G061 and the German Research Foundation (DFG) project KR 4890/1-1 "Verified Model Checkers" (317422601). A part of the frequency extension has been implemented within Google Summer of Code 2016.

$$\begin{tikzcd}
 & \text{DGRA} \arrow[r, "{[KE12]}"] & \text{DRA} \arrow[dr, "{[KMWW17]}"] & \\
\text{LTL} \arrow[ur, bend left, "{[EK14]}"] \arrow[r, dotted, "{[VW86] etc.}"] \arrow[dr, bend right, "{[SEJK16]}"'] \arrow[drr] & \text{NBA} \arrow[ur, dotted, "{[Saf88]}"] \arrow[rr, dotted, "{[Pit06,Sch09]}"] \arrow[dr, dotted, "{[CY88]}"] & & \text{DPA} \\
 & \text{LDGBA} \arrow[r, dotted] & \text{LDBA} \arrow[ur, bend right, "{[EKRS17]}"'] &
\end{tikzcd}$$

Fig. 1. LTL translations to different types of automata. Translations implemented in Rabinizer 4 are indicated with a solid line. The traditional approaches are depicted as dotted arrows. The determinization of NBA to DRA is implemented in ltl2dstar [Kle], to LDBA in Seminator [BDK+17] and to (mostly) DPA in spot [DLLF+16].

LTL. However, other types of automata, such as deterministic Muller and deterministic parity automata (DPA) are typically larger than DGRA in terms of acceptance condition or the state space, respectively.[1] Recently, several approaches with specific LDBA were proved applicable to the quantitative setting [HLS+15,SEJK16] and competitive with DGRA. Besides, model checking MDP against LTL properties involving frequency operators [BDL12] also allows for an automata-theoretic approach, via deterministic generalized Rabin mean-payoff automata (DGRMA) [FKK15].

LTL synthesis can also be solved using the automata-theoretic approach. Although DRA and DGRA transformed into games can be used here, the algorithms for the resulting Rabin games [PP06] are not very efficient in practice. In contrast, DPA may be larger, but in this setting they are the automata of choice due to the good practical performance of parity-game solvers [FL09,ML16,JBB+17].

Types of translations. The translations of LTL to NBA, e.g., [VW86], are typically *"semantic"* in the sense that each state is given by a set of logical formulae and the language of the state can be captured in terms of semantics of these formulae. In contrast, the determinization of Safra [Saf88] or its improvements [Pit06,Sch09,TD14,FL15] are not "semantic" in the sense that they ignore the structure and produce trees as the new states that, however, lack the logical interpretation. As a result, if we apply Safra's determinization on semantically created NBA, we obtain DRA that lack the structure and, moreover, are unnecessarily large since the construction cannot utilize the original structure. In contrast, the recent works [KE12,KLG13,EK14,KV15,SEJK16,EKRS17,MS17,KV17] provide "semantic" constructions, often producing smaller automata. Furthermore, various transformations such as degeneralization [KE12], index appearance record [KMWW17] or determinization of limit-deterministic automata [EKRS17] preserve the semantic description, allowing for further optimizations of the resulting automata.

Our contribution. While all previous versions of Rabinizer [GKE12,KLG13,KK14] featured only the translation LTL→DGRA→DRA, Rabinizer 4 now implements all the translations depicted by the solid arrows in Fig. 1. It improves all these

[1] Note that every DGRA can be written as a Muller automaton on the same state space with an exponentially-sized acceptance condition, and DPA are a special case of DRA and thus DGRA.

translations, both algorithmically and implementation-wise, and moreover, features the first implementation of the translation of a frequency extension of LTL [FKK15].

Further, in order to utilize the resulting automata for verification, we provide our own distribution[2] of the PRISM model checker [KNP11], which allows for model checking MDP against LTL using not only DRA and DGRA, but also using LDBA and against frequency LTL using DGRMA. Finally, the tool can turn the produced DPA into parity games between the players with input and output variables. Therefore, when linked to parity-game solvers, Rabinizer 4 can be also used for LTL synthesis.

Rabinizer 4 is freely available at `http://rabinizer.model.in.tum.de` together with an on-line demo, visualization, usage instructions and examples.

2 Functionality

We recall that the previous version Rabinizer 3 has the following functionality:

- It translates LTL formulae into equivalent DGRA or DRA.
- It is linked to PRISM, allowing for probabilistic verification using DGRA (previously PRISM could only use DRA).

2.1 Translations

Rabinizer 4 inputs formulae of LTL and outputs automata in the standard HOA format [BBD$^+$15], which is used, e.g., as the input format in PRISM. Automata in the HOA format can be directly visualized, displaying the "semantic" description of the states. Rabinizer 4 features the following command-line tools for the respective translations depicted as the solid arrows in Fig. 1:

ltl2dgra and **ltl2dra** correspond to the original functionality of Rabinizer 3, i.e., they translate LTL (now with the extended syntax, including all common temporal operators) to DGRA and DRA [EK14], respectively.

ltl2ldgba and **ltl2ldba** translate LTL to LDGBA using the construction of [SEJK16] and to LDBA, respectively. The latter is our modification of the former, which produces smaller automata than chaining the former with the standard degeneralization.

ltl2dpa translates LTL to DPA using two modes:
- The default mode uses the translation to LDBA, followed by a LDBA-to-DPA determinization [EKRS17] specially tailored to LDBA with the "semantic" labelling of states, avoiding additional exponential blow-up of the resulting automaton.
- The alternative mode uses the translation to DRA, followed by our improvement of the index appearance record of [KMWW17].

fltl2dgrma translates the frequency extension of LTL$_{\setminus\mathbf{GU}}$, i.e. LTL$_{\setminus\mathbf{GU}}$ [KLG13] with $\mathbf{G}^{\sim\rho}$ operator[3], to DGRMA using the construction of [FKK15].

[2] Merging these features into the public release of PRISM as well as linking the new version of Rabinizer is subject to current collaboration with the authors of PRISM.

[3] The *frequential globally* construct [BDL12,BMM14] $\mathbf{G}^{\sim\rho}\varphi$ with $\sim\in\{\geq,>,\leq,<\}, \rho\in[0,1]$ intuitively means that the fraction of positions satisfying φ satisfies $\sim\rho$. Formally, the fraction on an infinite run is defined using the long-run average [BMM14].

2.2 Verification and synthesis

The resulting automata can be used for model checking probabilistic systems and for LTL synthesis. To this end, we provide our own distribution of the probabilistic model checker PRISM as well as a procedure transforming automata into games to be solved.

Model checking: PRISM distribution For model checking Markov chains and Markov decision processes, PRISM [KNP11] uses DRA and recently also more efficient DGRA [CGK13,KK14]. Our distribution, which links Rabinizer, additionally features model checking using the LDBA [SEJK16,SK16] that are created by our **ltl2ldba**.

Further, the distribution provides an implementation of frequency $\mathrm{LTL}_{\setminus \mathbf{GU}}$ model checking, using DGRMA. To the best of our knowledge, there are no other implemented procedures for logics with frequency. Here, techniques of linear programming for multi-dimensional mean-payoff satisfaction [CKK15] and the model-checking procedure of [FKK15] are implemented and applied.

Synthesis: Games The automata-theoretic approach to LTL synthesis requires to transform the LTL formula into a game of the input and output players. We provide this transformer and thus an end-to-end LTL synthesis solution, provided a respective game solver is linked. Since current solutions to Rabin games are not very efficient we implemented a transformation of DPA into parity games and a serialization to the format of PG Solver [FL09]. Due to the explicit serialization, we foresee the main use in quick prototyping.

3 Optimizations, Implementation, and Evaluation

Compared to the theoretical constructions and previous implementations, there are numerous improvements, heuristics, and engineering enhancements. We evaluate the improvements both in terms of the size of the resulting automaton as well as the running time. When comparing with respect to the original Rabinizer functionality, we compare our implementation **ltl2dgra** to the previous version Rabinizer 3.1, which is already a significantly faster [EKS16] re-implementation of the official release Rabinizer 3 [KK14]. All of the benchmarks have been executed on a host with i7-4700MQ CPU (4x2.4 GHz), running Linux 4.9.0-5-amd64 and the Oracle JRE 9.0.4+11 JVM. Due to the start-up time of JVM, all times below 2 seconds are denoted by <2 and not specified more precisely. All experiments were given a time-out of 900 seconds and mem-out of 4GB, denoted by $-$.

Algorithmic improvements and heuristics for each of the translations:

ltl2dgra and **ltl2dra** These translations create a master automaton monitoring the satisfaction of the given formula and a dedicated slave automaton for each subformula of the form $\mathbf{G}\psi$ [EK14]. We (i) simplify several classes of slaves and (ii) "suspend" (in the spirit of [BBDL^{+}13]) some so that they appear in the final product only in some states. The effect on the size of the state space is illustrated in Table 1 on a nested formula. Further, (iii) the acceptance condition is considered separately for each strongly connected component (SCC) and then

Table 1. Effect of simplifications and suspension for **ltl2dgra** on the formulae $\psi_i = \mathbf{G}\phi_i$ where $\phi_1 = a_1, \phi(i) = (a_i\mathbf{U}(\mathbf{X}\phi_{i-1}))$, and $\psi'_i = \mathbf{G}\phi'_i$ where $\phi'_1 = a_1$, $\phi'_1 = (\phi'_{i-1}\mathbf{U}(\mathbf{X}^i a_i)$, displaying execution time in seconds / #states.

	ψ_2	ψ_3	ψ_4	ψ_5	ψ_6
Rabinizer 3.1 [EKS16]	<2 / 4	<2 / 16	<2 / 73	3 / 332	60 / 1463
ltl2dgra	<2 / 3	<2 / 7	<2 / 35	3 / 199	13 / 1155
	ψ'_2	ψ'_3	ψ'_4	ψ'_5	ψ'_6
Rabinizer 3.1 [EKS16]	<2 / 4	<2 / 16	2 / 104	128 / 670	–
ltl2dgra	<2 / 3	<2 / 10	<2 / 38	7 / 175	239 / 1330

Table 2. Effect of computing acceptance sets per SCC on formulae $\psi_1 = x_1 \wedge \phi_1$, $\psi_2 = (x_1 \wedge \phi_1) \vee (\neg x_1 \wedge \phi_2)$, $\psi_3 = (x_1 \wedge x_2 \wedge \phi_1) \vee (\neg x_1 \wedge x_2 \wedge \phi_2) \vee (x_1 \wedge \neg x_2 \wedge \phi_3)$, ..., where $\phi_i = \mathbf{XG}((a_i\mathbf{U}b_i) \vee (c_i\mathbf{U}d_i))$, displaying execution time in seconds / #acceptance sets.

	ψ_1	ψ_2	ψ_3	ψ_4	ψ_5	...	ψ_8
Rabinizer 3.1 [EKS16]	<2 / 2	<2 / 7	<2 / 19	–	–		–
ltl2dgra	<2 / 1	<2 / 1	<2 / 1	<2 / 1	<2 / 1		<2 / 1

Table 3. Effect of break-point elimination for **ltl2ldba** on safety formulae $s(n, m) = \bigwedge_{i=1}^{n} \mathbf{G}(a_i \vee \mathbf{X}^m b_i)$ and for **ltl2ldgba** on liveness formulae $l(n, m) = \bigwedge_{i=1}^{n} \mathbf{GF}(a_i \wedge \mathbf{X}^m b_i)$, displaying #states (#Büchi conditions)

	$s(1,3)$	$s(2,3)$	$s(3,3)$	$s(4,3)$	$s(1,4)$	$s(2,4)$	$s(3,4)$	$s(4,4)$
[SEJK16]	20 (1)	400 (2)	$8 \cdot 10^3$(3)	$16 \cdot 10^4$(4)	48 (1)	2304 (2)	110592 (3)	–
ltl2ldba	8 (1)	64 (1)	512 (1)	4096 (1)	16 (1)	256 (1)	4096 (1)	65536 (1)

	$l(1,1)$	$l(2,1)$	$l(3,1)$	$l(4,1)$	$l(1,4)$	$l(2,4)$	$l(3,4)$	$l(4,4)$
[SEJK16]	3 (1)	9 (2)	27 (3)	81 (4)	10 (1)	100 (2)	10^3 (3)	10^4 (4)
ltl2ldgba	3 (1)	5 (2)	9 (3)	17 (4)	3 (1)	5 (2)	9 (3)	17 (4)

Table 4. Effect of non-determinism of the initial component for **ltl2ldba** on formulae $f(i) = \mathbf{F}(a \wedge \mathbf{X}^i\mathbf{G}b)$, displaying #states (#Büchi conditions)

	$f(1)$	$f(2)$	$f(3)$	$f(4)$	$f(5)$	$f(6)$
[SEJK16]	4 (1)	6 (1)	10 (1)	18 (1)	34 (1)	66 (1)
ltl2ldba	2 (1)	3 (1)	4 (1)	5 (1)	6 (1)	7 (1)

combined. On a concrete example of Table 2, the automaton for $i = 8$ has 31 atomic propositions, whereas the number of atomic propositions relevant in each component of the master automaton is constant, which we utilize and thus improve performance on this family both in terms of size and time.

ltl2ldba This translation is based on breakpoints for subformulae of the form $\mathbf{G}\psi$. We provide a heuristic that avoids breakpoints when ψ is a safety or co-safety subformula, see Table 3.

Besides, we add an option to generate a non-deterministic initial component for the LDBA instead of a deterministic one. Although the LDBA is then no more suitable for quantitative probabilistic model checking, it still is for qualitative model checking. At the same time, it can be much smaller, see Table 4 which shows a significant improvement on the particular formula.

Table 5. Comparison of the average performance with the previous version of Rabinizer. The statistics are taken over a set of 200 standard formulae [KMS18] used, e.g., in [BKS13,EKS16], run in a batch mode for both tools to eliminate the effect of the JVM start-up overhead.

Tool	Avg # states	Avg # acc. sets	Avg runtime
Rabinizer 3.1 [EKS16]	6.3	6.7	0.23
ltl2dgra	6.2	4.4	0.12

ltl2dpa Both modes inherit the improvements of the respective **ltl2ldba** and **ltl2dgra** translations. Further, since complementing DPA is trivial, we can run in parallel both the translation of the input formula and of its negation, returning the smaller of the two results. Finally, we introduce several heuristics to optimize the treatment of safety subformulae of the input formula.

dra2dpa The index appearance record of [KMWW17] keeps track of a permutation (ordering) of Rabin pairs. To do so, all ties between pairs have to be resolved. In our implementation, we keep a pre-order instead, where irrelevant ties are not resolved. Consequently, it cannot happen that an irrelevant tie is resolved in two different ways like in [KMWW17], thus effectively merging such states.

Implementation The main performance bottleneck of the older implementations is that explicit data structures for the transition system are not efficient for larger alphabets. To this end, Rabinizer 3.1 provided symbolic (BDD) representation of states and edge labels. On the top, Rabinizer 4 represents the transition function symbolically, too.

Besides, there are further engineering improvements on issues such as storing the acceptance condition only as a local edge labelling, caching, data-structure overheads, SCC-based divide-and-conquer constructions, or the introduction of parallelization for batch inputs.

Average performance evaluation We have already illustrated the improvements on several hand-crafted families of formulae. In Tables 1 and 2 we have even seen the respective running-time speed-ups. As the basis for the overall evaluation of the improvements, we use some established datasets from literature, see [KMS18], altogether two hundred formulae. The results in Table 5 indicate that the performance improved also on average among the more realistic formulae.

4 Conclusion

We have presented Rabinizer 4, a tool set to translate LTL to various deterministic automata and to use them in probabilistic model checking and in synthesis. The tool set extends the previous functionality of Rabinizer, improves on previous translations, and also gives the very first implementations of frequency LTL translation as well as model checking. Finally, the tool set is also more user-friendly due to richer input syntax, its connection to PRISM and PG Solver, and the on-line version with direct visualization, which can be found at `http://rabinizer.in.tum.de`.

References

[BBD+15] Tomáš Babiak, František Blahoudek, Alexandre Duret-Lutz, Joachim Klein, Jan Křetínský, David Müller, David Parker, and Jan Strejček. The hanoi omega-automata format. In *CAV, Part I*, pages 479–486, 2015.

[BBDL$^+$13] Tomáš Babiak, Thomas Badie, Alexandre Duret-Lutz, Mojmír Křetínský, and Jan Strejček. Compositional approach to suspension and other improvements to LTL translation. In *SPIN*, pages 81–98, 2013.

[BDK$^+$17] František Blahoudek, Alexandre Duret-Lutz, Mikuláš Klokočka, Mojmír Křetínský, and Jan Strejček. Seminator: A tool for semi-determinization of omega-automata. In *LPAR*, pages 356–367, 2017.

[BDL12] Benedikt Bollig, Normann Decker, and Martin Leucker. Frequency linear-time temporal logic. In *TASE*, pages 85–92, 2012.

[BKŘS12] Tomáš Babiak, Mojmír Křetínský, Vojtěch Řehák, and Jan Strejček. LTL to Büchi automata translation: Fast and more deterministic. In *TACAS*, pages 95–109, 2012.

[BKS13] Frantisek Blahoudek, Mojmír Křetínský, and Jan Strejček. Comparison of LTL to deterministic Rabin automata translators. In *LPAR*, volume 8312 of *LNCS*, pages 164–172, 2013.

[BMM14] Patricia Bouyer, Nicolas Markey, and Raj Mohan Matteplackel. Averaging in LTL. In *CONCUR*, pages 266–280, 2014.

[CGK13] Krishnendu Chatterjee, Andreas Gaiser, and Jan Křetínský. Automata with generalized Rabin pairs for probabilistic model checking and LTL synthesis. In *CAV*, pages 559–575, 2013.

[CKK15] Krishnendu Chatterjee, Zuzana Komárková, and Jan Křetínský. Unifying two views on multiple mean-payoff objectives in markov decision processes. In *LICS*, pages 244–256, 2015.

[CY88] Costas Courcoubetis and Mihalis Yannakakis. Verifying temporal properties of finite-state probabilistic programs. In *FOCS*, pages 338–345, 1988.

[DLLF$^+$16] Alexandre Duret-Lutz, Alexandre Lewkowicz, Amaury Fauchille, Thibaud Michaud, Etienne Renault, and Laurent Xu. Spot 2.0 — a framework for LTL and ω-automata manipulation. In *ATVA*, pages 122–129, October 2016.

[EH00] Kousha Etessami and Gerard J. Holzmann. Optimizing Büchi automata. In *CONCUR*, pages 153–167, 2000.

[EK14] Javier Esparza and Jan Křetínský. From LTL to deterministic automata: A Safraless compositional approach. In *CAV*, pages 192–208, 2014.

[EKRS17] Javier Esparza, Jan Křetínský, Jean-Francois Raskin, and Salomon Sickert. From LTL and limit-deterministic Büchi automata to deterministic parity automata. In *TACAS*, pages 426–442, 2017.

[EKS16] Javier Esparza, Jan Kretínský, and Salomon Sickert. From LTL to deterministic automata - A safraless compositional approach. *Formal Methods in System Design*, 49(3):219–271, 2016.

[FKK15] Vojtěch Forejt, Jan Krčál, and Jan Křetínský. Controller synthesis for MDPs and frequency LTL\GU. In *LPAR*, pages 162–177, 2015.

[FL09] Oliver Friedmann and Martin Lange. Solving parity games in practice. In *ATVA*, pages 182–196, 2009.

[FL15] Dana Fisman and Yoad Lustig. A modular approach for büchi determinization. In *CONCUR*, pages 368–382, 2015.

[GKE12] Andreas Gaiser, Jan Křetínský, and Javier Esparza. Rabinizer: Small deterministic automata for LTL(F,G). In *ATVA*, pages 72–76, 2012.

[GL02] Dimitra Giannakopoulou and Flavio Lerda. From states to transitions: Improving translation of LTL formulae to Büchi automata. In *FORTE*, pages 308–326, 2002.

[GO01] Paul Gastin and Denis Oddoux. Fast LTL to Büchi automata translation. In *CAV*, pages 53–65, 2001. Tool accessible at `http://www.lsv.ens-cachan.fr/~gastin/ltl2ba/`.

[HLS$^+$15] Ernst Moritz Hahn, Guangyuan Li, Sven Schewe, Andrea Turrini, and Lijun Zhang. Lazy probabilistic model checking without determinisation. In *CONCUR*, volume 42 of *LIPIcs*, pages 354–367, 2015.

[JBB$^+$17] Swen Jacobs, Nicolas Basset, Roderick Bloem, Romain Brenguier, Maximilien Colange, Peter Faymonville, Bernd Finkbeiner, Ayrat Khalimov, Felix Klein, Thibaud Michaud, Guillermo A. Pérez, Jean-François Raskin, Ocan Sankur, and Leander Tentrup. The 4th reactive synthesis competition (SYNTCOMP 2017): Benchmarks, participants & results. *CoRR*, abs/1711.11439, 2017.

[KE12] Jan Křetínský and Javier Esparza. Deterministic automata for the (F,G)-fragment of LTL. In *CAV*, volume 7358 of *LNCS*, pages 7–22, 2012.

[KK14] Zuzana Komárková and Jan Křetínský. Rabinizer 3: Safraless translation of LTL to small deterministic automata. In *ATVA*, volume 8837 of *LNCS*, pages 235–241, 2014.

[Kle] Joachim Klein. ltl2dstar - LTL to deterministic Streett and Rabin automata. `http://www.ltl2dstar.de/`.

[KLG13] Jan Křetínský and Ruslán Ledesma-Garza. Rabinizer 2: Small deterministic automata for LTL$\setminus$GU. In *ATVA*, pages 446–450, 2013.

[KMS18] Jan Křetínský, Tobias Meggendorfer, and Salomon Sickert. LTL Store: Repository of LTL formulae from literature and case studies. *CoRR*, abs/1804.xxxx, 2018.

[KMWW17] Jan Křetínský, Tobias Meggendorfer, Clara Waldmann, and Maximilian Weininger. Index appearance record for transforming rabin automata into parity automata. In *TACAS*, pages 443–460, 2017.

[KNP11] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *CAV*, pages 585–591, 2011.

[KV15] Dileep Kini and Mahesh Viswanathan. Limit deterministic and probabilistic automata for LTL $\setminus$ GU. In *TACAS*, pages 628–642, 2015.

[KV17] Dileep Kini and Mahesh Viswanathan. Optimal translation of LTL to limit deterministic automata. In *TACAS 2017*, 2017. To appear.

[ML16] Philipp J. Meyer and Michael Luttenberger. Solving mean-payoff games on the GPU. In *ATVA*, pages 262–267, 2016.

[MS17] David Müller and Salomon Sickert. LTL to deterministic Emerson-Lei automata. In *GandALF*, pages 180–194, 2017.

[Pit06] Nir Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. In *LICS*, pages 255–264, 2006.

[Pnu77] Amir Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57, 1977.

[PP06] Nir Piterman and Amir Pnueli. Faster solutions of Rabin and Streett games. In *LICS*, pages 275–284, 2006.

[Saf88] Shmuel Safra. On the complexity of omega-automata. In *FOCS*, pages 319–327, 1988.

[SB00] Fabio Somenzi and Roderick Bloem. Efficient Büchi automata from LTL formulae. In *CAV*, pages 248–263, 2000.

[Sch09] Sven Schewe. Tighter bounds for the determinisation of Büchi automata. In *FoSSaCS*, volume 5504 of *LNCS*, pages 167–181, 2009.

[SEJK16] Salomon Sickert, Javier Esparza, Stefan Jaax, and Jan Kretínský. Limit-deterministic büchi automata for linear temporal logic. In *CAV*, pages 312–332, 2016.

[SK16] Salomon Sickert and Jan Kretínský. Mochiba: Probabilistic LTL model checking using limit-deterministic büchi automata. In *ATVA*, pages 130–137, 2016.

[TD14] Cong Tian and Zhenhua Duan. Buchi determinization made tighter. Technical Report abs/1404.1436, arXiv.org, 2014.

[VW86] Moshe Y. Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *LICS*, pages 332–344, 1986.