CODE AND DESIGN SAFETY OF PROBABILISTIC SYSTEMS

Habilitation Thesis
(collection of articles)

by

Petr Novotný

Masaryk University
Faculty of Informatics

2022

ABSTRACT

This thesis, submitted as a part of habilitation proceedings of its author, presents the author's contribution to the areas of *code* and *design* safety of probabilistic systems. Code safety refers to the process of ensuring that a program code involving probabilistic instructions behaves according to the intent of the programmer. Design safety refers to designing probabilistic systems so that they behave in an inherently safe way.

In the domain of code safety, the thesis details the author's work on static analysis of probabilistic programs via *martingales,* a concept from probability theory. We show how martingales can be used to reason about termination, safety, and efficiency questions pertaining to probabilistic programs and how such reasoning can be performed in an automated way.

On the design safety front, we describe the author's work on the problem of *risk-constrained* probabilistic decision-making. The idea is to design decision-making algorithms for autonomous agents so that apart from the usual objective of maximising the agent's expected utility, the algorithm ensures a bounded level of *risk,* quantitatively expressed via a suitable risk metric.

The thesis is written in the form of a collection of articles (Part II) accompanied by an extensive commentary (Part I). The commentary is written so as to provide a high-level overview of the problems tackled and the ideas behind their solutions. The technical details are contained in the enclosed articles.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank all my co-authors (and co-authors to be), whether the outcomes of our collaboration are included in this thesis or not. Namely, my thanks go to (in alphabetic order) Sheshansh Agrawal, Michal Ajdarów, Guy Avni, Nikhil Balaji, František Blahoudek, Tomáš Brázdil, Šimon Brlej, Krishnendu Chatterjee, Taolue Chen, Martin Chmelík, Murat Cubuktepe, Adrián Elgyütt, Vojtěch Forejt, Hongfei Fu, Ehsan Kafshdar Goharshady, Anchit Gupta, Rouzbeh Hasheminezhad, Thomas A. Henzinger, Rasmus Ibsen-Jensen, Deep Karkhanis, Joost-Pieter Katoen, Stefan Kiefer, David Klaška, Ľuboš Korenčiak, Jan Krčál, Antonín Kučera, Vít Musil, Melkior Ornik, Guillermo A. Pérez, Jean-François Raskin, Owen Rouillé, Amélie Royer, Vojtěch Řehák, Aistis Simaitis, Mahsa Shirmohammadi, Ufuk Topcu, Pranay Thangeda, Jiří Vahala, Dominik Velan, Dominik Wojtczak, Florian Zuleger, Jiří Zárevúcky, and Đorđe Žikelić. Of course, there were many more colleagues who have impacted my research career through discussions, suggestions, and friendships. It would be difficult to compile their list while avoiding the risk of inadvertently omitting someone, and hence I extend my thanks to all of you anonymously.

Special thanks go to Dana Komárková and Jan Strejček for helping with the administrative aspects of the habilitation procedure and to Benjamin Kaminski for helping in my search for nice thesis fonts.

Finally and most importantly, I would like to thank my wife Jana for her love and support and our son Vilém for bringing so much fun to our lives.

Petr Novotný
November 5, 2022

CONTENTS

Part I

COMMENTARY ON THE ENCLOSED PUBLICATIONS

PROBABILISTIC SYSTEMS IN COMPUTER SCIENCE

1

Probability?

Many people (including, from the author's experience, a non-trivial amount of computer scientists) view computer systems as an epitome of determinism. From this viewpoint, a computer ought to predictably follow the instructions set by the programmer, and any variation in its behaviour is deemed to be a result of errors in programming and of inadequate testing. While such a perception might be correct within *some* application domains, it is not accurate in general. Uncertainty and randomness pervade various fields of computing. In what might come as a surprise, this is not entirely the result of the recent rapid developments in artificial intelligence (though AI definitely *is* one of the major drivers for the surge of interest in probabilistic techniques); randomness is connected to the very beginnings of computer science as an independent field. Indeed, one of the first computations performed on the first general-purpose computer ENIAC were randomised[1] simulations of particle-level physical processes [Met87]. This concrete application nicely illustrates the typical usage of randomness in computing (similar to its usage in human reasoning): a computation that aims to say something about the real world necessarily operates over some model of the real world, and this model is necessarily imprecise; probabilities must then be used within the model to represent our uncertainty about the model/real-world correspondence. Domains using probabilistic methods include the aforementioned artificial intelligence (itself consisting of a diverse variety of sub-domains [SB18]), control theory [Kus71], planning and scheduling [LMA98], robotics [TBF05], simulation of complex systems (from weather and climate to epidemiological models [All08]), or security [MVV18].

Formal Reasoning about Probabilistic Computations

The proliferation of probabilistic reasoning in computer science quite naturally spawned a demand for *formal analysis* of probabilistic computations. Major developments in formal methods for probabilistic systems started to appear in the 1980s, mostly in the form of logical calculi for temporal properties of probabilistic programs [FH82; Fel84; Koz83]. This was followed by a strive towards *automation* of formal verification techniques for probabilistic systems, inspired by the success of automated verification in the *classical* (i.e. non-probabilistic) setting. Initially, the automation focused on finite-state probabilistic systems (Markov chains, Markov decision processes, and stochastic games), where algorithms verifying a wide range of properties were developed [BK08; KP12]. The logical next step was considering automated verification of decidable infinite-state program models,

typically probabilistic pushdown automata [BEKK12; EY12], and their restrictions, such as probabilistic BPAs [BBKO10] or one-counter automata [BBEK11; BKNW12; BBEKW10; BBE10; SEY13]. It is not surprising, given the inherent complexity of probabilistic systems, that the decidability frontier approached significantly faster with increasing model expressiveness compared to the classical setting. This became particularly evident when considering models capturing multiple unbounded variables, such as probabilistic vector addition system with states (*pVASS*) [BKKNK14; BKKN15] or branching processes [ESY12; ESY17].

While the strive towards automation of probabilistic formal methods ultimately covered most of the commonly studied models, challenges along various fronts remained (and remain) to be resolved. We do not aim to provide their exhaustive list here. Instead, the next subsection focuses on those challenges appearing since the early-to-mid 2010s that became the focus of the author's research. Then, we will sketch how the author tackled these challenges and show the outline of the remainder of this thesis which contains a more detailed account of the author's contribution.

Challenges

The first challenge we consider is the lifting of probabilistic formal methods from the world of abstract and decidable models to the world of more concrete and Turing-complete *probabilistic programs.* That is, make a step towards *probabilistic program analysis (PPA).*[2] This challenge mimics a similar process in the non-probabilistic setting. There, many efficient (though necessarily incomplete) automated techniques for proving a variety of program properties were developed, despite the inherent undecidability of verification in Turing-complete programs [Tur38]. Such an assortment of techniques was not available for probabilistic programs, although the progress on the PPA front was tremendous in the past decade. Going back to the early 2010s, the pioneering steps towards automated PPA focused on safety properties, e.g. bounding the probability of assertion violation via techniques such as probabilistic CEGAR [HWZ08] or game-based abstraction [KNP06]. There were markedly fewer results concerning *liveness* properties, such as termination and its quantitative aspects (e.g. expected termination time).

The previous paragraph paints the picture of the first challenge this thesis deals with: developing new automated techniques for proving properties of Turing-complete probabilistic programs, with a (non-exclusive) focus on liveness properties. We call this challenge *code safety*, since the guiding goal here is to ensure that a probabilistic code behaves correctly and according to the intents of its programmer.

The second challenge we consider focuses on what happens before we even start to check the correctness of a probabilistic program: the process of *designing* a probabilistic program so that it operates in a safe way. Hence, we call this challenge *design safety.* In this thesis, we study the design safety within the AI domain of *planning* for autonomous agents under *probabilistic uncertainty.* This domain has a rich history and a lot of common ground

with probabilistic formal methods: both fields employ similar models, such as Markov decision processes, and the methods developed within the field are often close in both intent and technical detail. (For instance, algorithms like value and policy iteration [Put05] are used within both fields.) However, agent planning puts more emphasis on *scalability* to large models, as opposed to the focus on theoretical guarantees sought in formal methods (e.g. a guarantee to find an optimal policy for the agent). This led to the development of many heuristic and approximate approaches to agent planning, including algorithms utilising machine learning techniques [SB18]. The heuristic nature of the approaches increases the risk that the agent will not operate according to the designer's intent. This risk is compounded by the fact that agent planning algorithms typically optimise the agent's *mean* performance, expressed in the form of an expected reward accumulated along the agent's run. The focus on expected value might allow for large and potentially dangerous deviations from the mean behaviour. The design safety challenge, as understood in this thesis, deals with designing agent planning algorithms that combine the efficiency of the heuristic approaches with ideas from formal methods to mitigate the aforementioned risk.

Author's Contributions

This thesis presents its author's contributions to tackling the *code* and *design* safety challenges for probabilistic systems.

Static Analysis of Probabilistic Programs with Martingales Chapters 3–5 focus on code safety, in particular on analysis of termination properties in probabilistic programs. The primary focus will be on *almost-sure (a.s.)* termination, i.e. proving that a program terminates with probability 1. We will show how the mathematical concept of *martingales* from probability theory has been leveraged for automation of a.s. termination proving. Chapter 4 reviews the conceptual roots of the approach present in the previous work on ranking functions and scalar martingales, and it (or parts thereof) can be safely skipped by a reader acquainted with the topic. Chapter 5 presents the author's contribution to martingale-based program analysis techniques.

Risk-Aware Agent Planning Chapter 6 focuses on design safety. We will present new agent planning algorithms that use ideas from constrained optimisation to mitigate the planning risk. That is, we assign to each policy some measure of *risk*, typically expressed as the probability of either entering some undesirable state (*state-based* risk) or of the reward dropping below some acceptable level (*tail* or *quantile* risk). We then seek policies optimising the expected payoff subject to the constraint that the risk in question is below a given threshold T. To tackle this problem, our algorithms combine ideas from heuristic search and reinforcement learning with tools inspired by the work on constrained and multi-objective Markov decision processes.

The thesis is written in the form of a collection of papers (Part II) provided with an extensive commentary (Part I). We aim to provide a high-level overview of the author's conceptual contribution to the above topics. The technical details can be found in the referenced publications.

PRELIMINARIES

2

2.1 Basic Notation

We assume familiarity with the basics of probability theory [Wil91]. In particular, a *probability space* is a triple $(\Omega, \mathcal{F}, \mathbb{P})$, where Ω is a *sample space*, $\mathcal{F}$ is a *sigma-algebra* of measurable sets over Ω, and $\mathbb{P}$ is a *probability measure* on $\mathcal{F}$. A *random variable (r.v.)* $R : \Omega \to \mathbb{R} \cup \{\pm\infty\}$ is an $\mathcal{F}$*-measurable* real-valued function (i.e. $\{\omega \mid R(\omega) \leq x\} \in \mathcal{F}$ for all $x \in \mathbb{R}$) and we denote by $\mathbb{E}[R]$ its *expected value*. We will also use random variables of the form $R \colon \Omega \to A$ for some finite or countable set A, which easily translates to the real-valued variables.

2.2 Modelling Probabilistic Computations

Before we delve into the world of probabilistic program analysis, we fix some necessary notation and define concepts that will serve as our mathematical "modelling language" throughout the thesis. Chief of these is the notion of a *probabilistic transition system.*

Definition 1 (Probabilistic transition system)**.** *A* probabilistic transition system (PTS) *is a tuple* $\mathcal{T} = (S, \Delta)$ *where:*

- S *is a set of* states*; and*
- Δ *is a set of* transitions*. Here, a transition is a tuple* $\tau = (s, \overrightarrow{\tau})$ *where* $s \in S$ *is a* source state *and* $\overrightarrow{\tau}$ *is a* next state *function specifying how probable it is for* $\mathcal{T}$ *to transition from* s *into each particular state when executing* τ*. Formally,* $\overrightarrow{\tau}$ *is either a probability distribution over* S *(if* S *is discrete) or a Markov kernel [MT09] of type* $S \times S \to [0, 1]$ *(if* S *is continuous).*

We say that a transition τ *is* enabled *in a state* s *if* s *is the first component (the source state) of* τ*. We denote by* $\Delta(s)$ *the set of all transitions enabled in* s*. We stipulate that* $\Delta(s) \neq \emptyset$ *for each state* s*.*

In this thesis, we use PTSs as an abstract "back-end" through which we define the semantics of various probabilistic models, including Turing-complete probabilistic programs. For now, we consider a coarse-grained classification of PTSs into three classes:

- *finite* PTSs, where both the sets S and Δ are finite,
- *discrete* PTSs, where S and Δ are finite or countably infinite sets (*discrete* subsumes *finite*),

- *continuous* or *general state space* PTSs, where S is an arbitrary set.

Below, we sketch the *semantics* of PTSs: we will describe how each PTS encodes a family of stochastic processes. We will keep the description on an intuitive level, using a language suitable mostly for discrete PTSs. The presented intuition also applies to continuous PTSs, though formalising it in the continuous setting would require delving into technicalities beyond the scope of the thesis. We refer the reader to the literature on general state space stochastic processes (e.g. [MT09]) for further details.

Semantics of PTSs

To define the dynamics of a PTS, an *initial state* s_0 and a *scheduler* σ have to be fixed. Here, a scheduler is a function which resolves the nondeterminism present in the system due to the fact that multiple (indeed, infinitely many) transitions can be enabled in some states. Formally, each scheduler is a function which takes as an input the whole current history of states and transitions exhibited by the PTS (i.e. the sequence $s_0\tau_0 s_1\tau_1 \dots s_{i-1}\tau_{i-1}s_i$, where s_i and τ_i are the i-th state and transition encountered so far, respectively) and outputs[3] a distribution over $\Delta(s_i)$.

Our PTSs evolve in *discrete time steps.* We start in the initial state s_0. Then, in each step i, the PTS is in some state $s_i \in S$ and performs a transition $\tau_i \in \Delta(s_i)$ sampled according to $\sigma(s_0\tau_0 s_1\tau_1 \dots s_{i-1}\tau_{i-1}s_i)$. Once this is done, the next state s_{i+1} is sampled according to $\overrightarrow{\tau_i}$. This process continues *ad infinitum* and thus (randomly) produces a single *trajectory* (or *run*) $s_0\tau_0 s_1\tau_1 s_2\tau_2 \dots$.

Thus, an intuitive view of the semantics of a PTS is that of an abstract "parameterised sampler" which inputs an initial state s_0 and a measurable scheduler σ and randomly samples an infinite state-transition trajectory according to the aforementioned rules. More precisely, a PTS $\mathcal{T}$ induces a function which, given s_0 and σ, outputs a probability measure $\mathbb{P}^{\sigma}_{\mathcal{T},s_0}$ over $\mathcal{T}$'s runs. The measure captures the behaviour of $\mathcal{T}$ under σ, sketched in the previous paragraph. We denote by $\mathbb{E}^{\sigma}_{\mathcal{T},s_0}$ the expected value operator associated with $\mathbb{P}^{\sigma}_{\mathcal{T},s_0}$. We will omit $\mathcal{T}$ from the subscript when clear from the context.

Markov Decision Processes and Stochastic Games. A PTS can be viewed as a type of a Markov decision process (MDP) [Put94] with a possibly general state space. A possible extension of the model gives rise to *stochastic games* that contain two types of nondeterminism. Formally, we can partition the set of states S into sets S_A, S_D of *angelic* and *demonic* states, respectively. Two schedulers, angelic one σ_A and demonic one σ_D are then involved in the resolution of nondeterminism: σ_A chooses transitions in angelic states, while σ_D in demonic ones. In the context of verification, σ_A typically models the behaviour of a rational agent who wants the system to achieve a given specification, while σ_D (which typically models unknown aspects of the system) aims to hinder such achievement. In this work, we will make only passing reference to stochastic games and hence omit the details. In the chapters on verification, we will take the demonic view of non-determinism, which

amounts to overapproximating the possible behaviours of the system. On the other hand, in chapters focused on probabilistic planning, we will take the angelic view, since we will use PTSs to model autonomous agents operating towards some desired goal.

INTRODUCTION TO CODE SAFETY

3

In this chapter, we describe concrete problems in probabilistic program analysis that we want to address. The following chapters describe the techniques used to solve these problems. These descriptions will, for each studied problem, consist of two parts:

- *Proof rule & certificate:* First, we will formulate a mathematical proof rule for proving the property addressed in the studied problem, e.g. almost-sure termination. The proof rule will typically be of the form "if there exists a *proof certificate* of a specific form, then the property holds." For instance, in the case of almost-sure termination, we will show that if a program admits a so-called *ranking supermartingale,* then it terminates almost-surely. The proof rules and certificates are abstract objects; thus, they will be defined and analysed (w.r.t. soundness and completeness) over the abstract model of PTSs. In particular, our proof rules will be applicable to any probabilistic system that can be modelled as a PTS, not only to models stemming from a concrete form of probabilistic programs.
- *Automation:* Once we define a sound rule for some property, we focus on its *automation,* typically in the form of designing an algorithm capable of synthesising the respective proof certificate. To do this, we need to use a more concrete program model than PTSs: indeed, for infinite-state PTSs it is not even clear how to represent them in a finitary way so as to submit them to algorithmic analysis. In our work, we focus on algorithmic analysis of *imperative arithmetic programs*. Thus, whenever considering the question of automation, we will focus on PTSs representable by such programs. The syntax and semantics of this class of programs are provided in Section 3.2.

3.1 Problems in Analysis of Probabilistic Programs

We focus on proving the fundamental temporal properties of probabilistic programs listed below. We frame the properties in abstract terms, over a PTS $\mathcal{T} = (S, \Delta)$.

Almost-sure termination: We designate sets $In, F \subseteq S$ of *initial* and *terminal* states, respectively, and say that $\mathcal{T}$ is *almost-surely (a.s.) terminating* if for every scheduler σ and every initial state $s_0 \in In$ it holds that

$$\mathbb{P}^{\sigma}_{s_0}[\text{a state from } F \text{ is reached}] = 1.$$

The task is to prove that a given program is a.s. terminating.

Quantitative termination: the problem setup is the same as for a.s. termination, but the task is to compute an approximation of or bounds on the worst-case termination probability, i.e. the quantity

$$\inf_{\sigma} \inf_{s_0 \in In} \mathbb{P}^{\sigma}_{s_0}[\text{a state from } F \text{ is reached}].$$

Performing such an analysis makes sense when the program in question does not terminate a.s. or when its a.s. termination cannot be proved by available methods. Since we typically view termination as a positive event, we focus on computing lower bounds on its probability.

Finite termination: We say that a program is *finitely terminating* if it terminates in a finite expected number of steps. Each finitely terminating program is also a.s. terminating, since non-terminating runs have, by definition, an infinite termination time, and thus, their measure must be zero if the expected runtime is finite. The converse is not true. For instance, a program simulating a symmetric random walk with an absorbing barrier at 0 will terminate a.s., but its expected termination time is infinite.

Formally, given a target set F of states, we define a random variable $Time_F$ such that for each run ϱ the quantity $Time(\varrho)$ denotes the first point in time in which the run visits a state from F (or ∞ if no such time step exists). The variable $Time_F$ is typically called the *termination time.* Then, given a set of initial states In, the task is to decide whether for each $s_0 \in In$ and each scheduler σ it holds

$$\mathbb{E}^{\sigma}_{s_0}[Time_F] < \infty.$$

The property of finite termination is sometimes called *positive termination* in the literature, due to the analogy with the concept of positive recurrence in infinite-state Markov chains.

Expected termination time analysis: The setup is the same as in the finite termination problem, but the task is to compute some approximation of the worst-case expected termination time. Since we typically see early termination as preferable to later one, we usually aim to compute upper bounds (as tight as possible) on the quantity

$$\sup_{\sigma} \sup_{s_0 \in In} \mathbb{E}^{\sigma}_{s_0}[Time_F].$$

To obtain state-specific bounds, we often aim to compute a function which inputs s_0 and returns an upper bound on $\sup_{\sigma} \mathbb{E}^{\sigma}_{s_0}[Time_F]$. For arithmetic programs where the state is characterised by valuation of the program's variables, such a bounding function typically takes form of a symbolic expression involving these variables.

Tail bounds on runtime: The expected value is a rather crude characteristic of a random variable, and the expected termination time is no exception to this fact. A program might have a small expected runtime, but the probability of extremely high runtimes could still be relatively high: the runtime of a probabilistic program could follow a *fat-tailed* distribution. To account for such eventualities, we might want to compute finer characteristics of the

runtime, such as its tail bounds. That is, we want to compute functions $N_0 : S \to \mathbb{N}$ and $g : S \times \mathbb{N} \to [0, 1]$ (both typically in the form of a symbolic expression) such that for every initial state s_0 and every $n \geq N_0(s_0)$ it holds

$$\sup_{\sigma} \mathbb{P}^{\sigma}_{s_0}[Time_F \geq n] \leq g(s_0, n).$$

Of particular interest are *exponentially decreasing* tail bounds, where $g(s_0, n)$ converges exponentially fast to 0 as $n \to \infty$. Exponentially decreasing tail bounds entail that the distribution of the runtime is concentrated around its expected value (the scale of the concentration being determined by constants appearing in the definitions of N_0 and g).

Quantitative safety: The above problems focus on various aspects of termination (or in general, *reachability*), which is a liveness property. It is natural to also study *safety* properties, where we aim to bound (from above) the probabilities of some undesirable events. The basic safety property is *state safety,* where the undesirable event is specified as reaching some set of undesirable states. In the context of imperative programs, we speak about the probability of *assertion violation,* since the undesirable states are typically those where the variable valuation violates some assertion in the program. Formally, given a set $V \subseteq S$ of violating states and an initial state s_0, we want to compute an upper bound (again, as tight as possible) on

$$\sup_{\sigma} \mathbb{P}^{\sigma}_{s_0}[\text{a state from } V \text{ is reached}].$$

As before, the bound can be either a concrete number, if s_0 is fixed in advance, or a symbolic expression determining a function from the set of initial states In to the unit interval.

3.2 Probabilistic Arithmetic Programs

While the problems we study and their theoretical solutions we provide are phrased in the language of abstract PTSs, for algorithmic solutions we restrict to a sub-class of PTSs that represent imperative arithmetic programs. More concretely, we study a simple imperative programming language consisting of standard programming constructs (assignments, conditionals, while loops, and sequencing of statements) operating over numerical variables (mathematical reals, rationals, or integers).[4] Our language also contains two inherently *probabilistic* constructs: sampling of a variable value from some probability distribution (**sample** instruction) and probabilistic branching (the **if prob(*p*)** construct, which selects the if branch with probability p and the else branch with probability $1 - p$). Our programs also contain support for non-determinism: non-deterministic branching (**if** $\star$ statement, where the choice between the if and the else branch is resolved by a scheduler) and non-deterministic variable update (**ndet**), which sets a variable to a value non-deterministically chosen from a set of valuations satisfying a given expression. The abstract grammar of the programming language is summarised in Figure 1.

$$
\begin{aligned}
\langle stmt\rangle ::= {} & \langle assgn\rangle \mid \textbf{skip} \mid \langle stmt\rangle; \langle stmt\rangle \\
& \mid \textbf{if } \langle ndbexpr\rangle \textbf{ then } \langle stmt\rangle \textbf{ else } \langle stmt\rangle \textbf{ fi} \\
& \mid \textbf{while } \langle boolexpr\rangle \textbf{ do } \langle stmt\rangle \textbf{ od} \\
\langle assgn\rangle ::= {} & \langle var\rangle := \langle expr\rangle \\
& \mid \langle var\rangle := \textbf{sample(}\langle dist\rangle\textbf{)} \\
& \mid \langle var\rangle := \textbf{ndet(}\langle expr\rangle\textbf{)} \\
\langle ndbexpr\rangle ::= {} & \star \mid \textbf{prob(}\langle p\rangle\textbf{)} \mid \langle boolexpr\rangle
\end{aligned}
$$

```
while x ≥ 0 ∧ y ≥ 0 do
  if ∗ then
    if prob(1/2) then
      x := x − 2
    else skip
    fi
  else
    x := 2x ;
    y := y − 1
  fi
od
```

Figure 1: Left: Abstract grammar of probabilistic programs considered in this chapter. Here, $\langle var\rangle$ ranges over a countable set of program variables, $\langle expr\rangle$ over the set of arithmetic expressions, $\langle dist\rangle$ over probability distributions over integers or reals, and $\langle boolexpr\rangle$ ranges over the set of boolean expressions (boolean combinations of atomic predicates of the form $E_1 \leq E_1$, where E_1, E_2 are arithmetic expressions).
Right: An example probabilistic program.

We consider both discrete and continuous distributions, and we assume that the expected value and the support of each distribution used in the program are known and well-defined. As for expressions, for theoretical development we allow arbitrary *measurable* expressions, which are built from program variables and Borel-measurable functions over integers or real numbers. The measurability condition is, in particular, satisfied for all the standard arithmetic operations utilised in basic calculus.

Affine Probabilistic Programs When considering automation of the presented approaches, we restrict to a class of *affine probabilistic programs (APPs)*. In APPs, all arithmetic expressions are of the form $d + \sum_{i=1}^{n} c_i x_i$, where $c_1, \ldots, c_n, d$ are constants and $x_1, \ldots, x_n$ are program variables. This restriction will allow us to reduce the search for proof certificates to linear programming, thus allowing for efficient automation. We note, however, that many techniques presented in this thesis were recently extended also to programs with non-linear arithmetic [CFG16].

From Programs to PTSs A probabilistic program can be naturally represented by a PTS whose states are tuples (ℓ, ν), where ℓ is a program *location* (intuitively, corresponding to a logical line of code or a program command currently being executed) and ν is a valuation of program variables. The transitions of the PTS are defined so as to capture the control flow of the program. For instance, the command $\ell =$ "**if prob**(0.6) **then** C_1 **else** C_2" is represented by adding, for each state of the form (ℓ, ν), a transition τ whose source state is (ℓ, ν) and whose next state function assigns probability 0.6 to (ℓ_1, ν) and probability 0.4

to (ℓ_2, ν), where ℓ_i is the first command of the sub-program C_i. Similarly, the command ℓ = "x := **sample**(Uniform [0,1]); C" is represented by adding a transition τ from each state ℓ, ν, such that the next state function of τ is a Markov kernel representing a uniform distribution over states of the form $(\ell', \nu(x \leftarrow a))$ for some $a \in [0, 1]$ and for ℓ' being the first command of C. The remaining programming constructs are represented in a similar manner, as common in program analysis.

In the context of imperative programs, the set of terminal states F is typically a set of states of the form (ℓ_{term}, ν), where ℓ_{term} is a special terminal location representing the "last line of code" in which the program has terminated. Similarly, there is an initial location ℓ_{init} representing the first line of code. By default, any variable valuation is considered initial: when this is not the desired interpretation, an initial block of (possibly non-deterministic) assignments specifies the permitted range of initial values.

RANKING FUNCTIONS AND MARTINGALES FOR TERMINATION PROVING: A BRIEF HISTORY

4

While this chapter aims to survey previous martingale-based techniques for probabilistic program analysis, we start our survey in the non-probabilistic world. This might seem counter-intuitive: the notion of *martingale* is (as we shall see) decidedly probabilistic, and hence it seems unnecessary to tie it to non-probabilistic concepts. However, the story of martingales in probabilistic program analysis does not primarily seem to be the case of "technological transfer" from abstract probability theory to the domain of formal methods. Instead, the story can be interpreted as a quest for a generalisation of techniques (proof certificates and algorithms) successful in the non-probabilistic domain to the probabilistic one. In particular, the martingales we will encounter can be viewed as a "syntactically natural" generalisation of so-called *ranking functions* to the probabilistic world, in the sense that the definitions of the two concepts look syntactically very similar, apart from certain differences that seem natural when augmenting a non-probabilistic concept to deal with probabilities. We stress that "naturalness" does not equate "straightforwardness." While defining probabilistic generalisations of ranking functions is not difficult, proving their soundness as a proof certificate and utilising their full power for program analysis are endeavours that often require delicate probabilistic reasoning and knowledge of tools of martingale theory: in this way, the connection to abstract probability is eventually created. However, the connection to the original source of the probabilistic generalisation, i.e. to ranking functions, is still present and becomes important when focusing on *automation:* due to the syntactic similarity of the original and the probabilistic proof certificates, the algorithms computing the latter ones share the same design patterns as the algorithms for the former ones. Hence, the next section provides a brief survey of ranking functions and algorithms for their synthesis.

4.1 History of Ranking Functions

We will treat the term *non-probabilistic program* mostly informally. Where a more formal view is required, we can imagine arithmetic programs as defined in Section 3.2 (including their formalisation via PTS's) but without any probabilistic commands (i.e. no probabilistic branching or sampling of a variable value). The termination analysis for non-probabilistic

programs aims to (dis)prove that all runs (or the single run, if there is no non-determinism) of a given program do reach a terminal state (i.e., *terminate*).

Ranking Functions Informally, a *ranking function* (RF) is a mapping $f : S \to W$ from the set of states of a program to a *well-founded* set, i.e. an ordered set $(W, \preceq)$ in which there are no infinite strictly descending sequences. To call such a function f *ranking,* it must satisfy some sort of a *strict decrease* condition: in the most simple form, we require that the value of f strictly decreases (w.r.t. the ordering $\preceq$) with each step of the program's execution (i.e. until a terminal state is reached). If a program admits such a ranking function, it clearly has to terminate: a non-terminating run $s_0\tau_0 s_1\tau_1 s_2\tau_2 \ldots$ would induce an infinite decreasing sequence $f(s_0) \succ f(s_1) \succ f(s_2) \succ \ldots$ of values in W, contradicting the well-foundedness of this set. One can readily notice that requiring a decrease in every step of an imperative program's execution is not strictly necessary, since we only need to show that the program does not get stuck in an infinite loop. To this end, it suffices to identify a set of *cutpoints,* i.e. a set of program locations such that each program loop goes through at least one such cutpoint, and require that the value of f strictly decreases between every two visits of cutpoints. (One natural choice of cutpoints are the heads of each loop.) While the resulting notion of a ranking function is, from the theoretical point of view, no more powerful than the one with a strict decrease in every step, using cutpoints might simplify some termination proofs. Another possible relaxation of the RF definition is transferring the well-foundedness requirement from W to f itself: that is, we do no longer require that W is well-founded, but then f itself must satisfy some additional property which ensures that there cannot be an infinite sequence $f(s_0) \succ f(s_1) \succ f(s_2) \succ \ldots$ induced by a run in the program.

Scalar Ranking Functions There are several natural choices for the set W, the most prominent being non-negative integers with their natural ordering. For programs with real arithmetic, it might be beneficial to work with non-negative rationals or reals, in which case the strict decrease condition must be strengthened to require that the magnitude of the decrease is uniformly bounded from below (e.g. *"between every two visits of cutpoints, the value of f must decrease by at least 1"*). We refer to RFs with such co-domains as *scalar,* since they assign a single number to each program state.

Completeness of Ranking Functions We have already argued why ranking functions are sound for proving program termination. It turns out that for programs with bounded non-determinism (which means that there is some uniform bound B on the number of successors of every state with non-deterministic choice), they are also *complete* in the sense that every terminating program admits a (scalar) ranking function. This is not hard to see: by König's lemma [Kön27], the set of states visited by a terminating program with bounded non-determinism must be finite, and hence also the maximal termination time from each reachable state is finite. It is then easy to see that a function assigning to each

state its maximal (over all runs initiated in that state) termination time is a scalar ranking function with codomain $\mathbb{Z}$.

Vectorial and Lexicographic Ranking Functions While, as mentioned above, scalar RFs are (in theory) sufficient to prove termination of any terminating program, termination proofs are typically easier to construct when the co-domain is endowed with a richer structure. In practice, the most commonly used ranking functions are arguably the *lexicographic* ones (*LexRFs*), which map the states to vectors of numbers ordered lexicographically. The concrete co-domain of the LexRF f can be either directly well-founded (say $\mathbb{N}^n$ ordered lexicographically) or more general, with the well-foundedness being provided by the definition of f. As an example of the latter approach, consider a function f mapping S to $\mathbb{Z}^n$ equipped with a lexicographic ordering (obviously not a well-founded set) such that in each step of the program's execution:

- the value of f decreases in lexicographic ordering, and
- the leftmost component of f to decrease is non-negative before executing the step.

Here, even though the codomain of f is not well-founded, one can easily verify that a program admitting such a function must terminate.

One might wonder why are the LexRFs preferable over their scalar counterparts. One key reason is that LexRFs are apt for reasoning about programs with non-trivial control flow structure. In particular, termination of nested loops can be nicely captured by the lexicographic ordering, with the leftmost component typically ranking (i.e. proving the termination of) the outer loop and one additional component added for each level of nesting. Automated tools for ranking function synthesis benefit from this type of structural decomposition: trying to directly compute a scalar RF can be too ambitious (e.g. there is no guarantee that the function admits a nice closed-form representation).

The LexRFs can be viewed as a special case of *vectorial* ranking functions, called so (unsurprisingly) because their codomains are sets of vectors. Examples of vectorial RFs which are not lexicographic RFs are the *Ramsey*-based ranking functions presented in [PR04b] and [CPR06].

Brief History of Ranking Functions The termination analysis literature often cites the paper of Floyd [Flo67] as the originator of ranking functions. The paper lays out a general framework for program verification, stating an unpublished work of Alan Perlis and Saul Gorn as a source of inspiration. However, similar ideas about proving correctness and termination can be traced back even further, to the Alan Turing's 1949 three-page conference piece "Checking a Large Routine" [Tur49]. The likely reason for the relative obscurity of Turing's paper is, as stated in [MJ84], a large number of typos in its typewritten transcript, which obfuscates its message.

It is worth noting that the papers of both Turing and Floyd already consider what we previously called a lexicographic ranking function. Also, none of them actually uses the term *ranking function:* Floyd uses the name *W-function* (as a shorthand for ***w****ell-ordered*), while Turing does not use any specific name at all. The earliest use of the term *ranking function* the author was able to find is in the paper [LPS81], which deals with termination proving in concurrent programs. Other terms used for the concept of RFs in the past are *termination function* [DM79], and *loop variant,* which Gries [Gri03] attributes to Dijkstra [Dij76].

Ranking functions bear some resemblance to *Lyapunov functions* used in control theory to prove the stability of a dynamical system. This connection does not seem to be discussed in the termination proving literature, likely because of the rather continuous nature of Lyapunov functions as opposed to discrete reasoning used in program analysis (e.g. in Lyapunov functions, the decrease condition is replaced with a condition placed on the function's derivatives). The connection between control theory and termination proving seems to be more pronounced in the probabilistic setting. Indeed, in probability theory, the term Lyapunov function is used in the context of so-called Foster-Lyapunov criteria [Fos53] to prove positive recurrence in infinite-state Markov chains, including chains with discrete state space. Such Lyapunov functions can be seen as natural probabilistic generalisations of scalar ranking functions, and hence the "Lyapunov" terminology is used in some early literature on probabilistic termination proving. We will provide more details on this topic in the subsequent sections.

Linear RFs and Supporting Invariants While the aforementioned works did not consider automated termination proving, they often contain conceptual points that streamline the proving process and thus act as stepping stones towards the automation. The paper [KM75] highlights two such points: *linear RFs* and supporting *invariants.*

A (vectorial) RF f is linear, if at every (cutpoint) location, each component of f is an affine function of the current valuation ν of program variables. That is, denoting by f_k the k-th component of f, we have $f(\ell, \nu) = b^\ell + \sum_{i=1}^{n} a_i^\ell x_i$, where $a_1^\ell, \ldots, a_n^\ell, b^\ell$ are location-specific components and $x_1, \ldots, x_n$ are the program variables. Restricting to linear RFs makes constructing and checking termination proofs easier, which comes at the expense of coverage: even if a program admits a ranking function, there is no guarantee that it admits a linear one. This limitation can, to some degree, be mitigated by using vectorial RFs: there are programs that only admit non-linear scalar RFs, but do admit linear vectorial RFs. [ACN18].

Now one might ask whether the use of linear RFs makes sense: one of the key requirements on an RF is that it is in some sense bounded from below (well-foundedness). But non-constant linear functions are clearly not bounded, so how could they act as RFs? The key observation is that an RF does not have to be lower-bounded over all *possible* states but only over *reachable* states, i.e. states that can appear during the program's execution. A similar observation holds about the strict decrease condition.

Precisely identifying the set of all reachable states is generally difficult and not possible to automate due to undecidability barriers. However, to check that f is a ranking function, it is sufficient to check the required conditions on some *over-approximation* of the set of reachable states. Such over-approximations are called *invariants.* Formally, an invariant for a program is a mapping I assigning to each location a set of variable valuations such that each state (ℓ, v) reachable from the initial state satisfies $v \in I(\ell)$. For a state $s = (\ell, v)$, we will use $s \in I$ as a short-hand for $v \in I(\ell)$.

An invariant I *supports* an RF f if f satisfies the well-foundedness and strict decrease condition once its domain is restricted to $\bigcup_\ell \{\ell\} \times I(\ell)$. The ability to compute good invariants is an important prerequisite of many automated termination-proving techniques.

Algorithmic Synthesis of Ranking Functions: The Farkas' Approach The first results concerning the automated synthesis of linear scalar RFs started to appear in the early 2000s, e.g. [CS01; PR04a]. While these papers considered programs of various syntactic structures and their approaches provided different sorts of guarantees, they were all based on the *template-based* approach utilising the *Farkas' lemma.* We will sketch the approach below, since it naturally transfers to probabilistic termination proving.

The approach is most suitable for affine, i.e. linear-arithmetic programs. Suppose that we are given such a program and its invariant I. For simplicity, we make the following two assumptions

- The invariant I is *polyhedral,* i.e. each of the sets $I(\ell)$ is an intersection of half-spaces.
- Each location forms a cutpoint, i.e. we want to ensure the strict decrease and lower bound conditions in every location.

Since linear ranking functions can be arbitrarily re-scaled and shifted, we can w.l.o.g. restrict our search to ranking functions that are always *positive* and that decrease by at least one in every step. Since we aim to find a linear scalar RF, we can fix, for each location ℓ, a linear template

$$b^\ell + \sum_{i=1}^{n} a_i^\ell \cdot x_i,$$

where $x_1, \ldots, x_n$ are all the programs variables and $a_1^\ell, \ldots, a_n^\ell, b^\ell$ are *unknown* coefficients. Substituting concrete values for these unknown coefficients in every location yields a linear function mapping program states to reals and vice versa. Hence, we can identify RF candidates f with $|L| \cdot (n+1)$-dimensional vectors (L being the set of all locations).

Now a vector f defines a ranking function iff for every location ℓ and every location ℓ' reachable in one step from ℓ, the following two formulae $NNEG_\ell$ (for "non-negativity") and $DEC_{\ell,\ell'}$ (for "decrease") are satisfied:

$$NNEG_\ell(f) \equiv \forall \nu : \nu \in I(\ell) \implies b^\ell + \sum_{i=1}^{n} a_i^\ell \cdot \nu(x_i) > 0, \tag{1}$$

$$DEC_{\ell,\ell'}(f) \equiv \forall \nu : \nu \in I(\ell) \implies b^{\ell'} + \sum_{i=1}^{n} a_i^{\ell'} \cdot \nu'(x_i) < b^\ell + \sum_{i=1}^{n} a_i^\ell \cdot \nu(x_i) - 1, \tag{2}$$

where ν' is the new variable valuation obtained by executing the command corresponding to the step from location ℓ to ℓ' in valuation ν. A careful reader might be surprised by the strict inequalities in the consequents, since the previous discussion called only for non-strict ones. However, the change of non-strict inequalities to strict ones is w.l.o.g. again due to the possibility of re-scaling and shifting an RF, and will later simplify the construction.

Let us now investigate the syntactic structure of the two types of formulae. The consequent in $NNEG_\ell$ is a (strict) linear inequality (linear w.r.t. the terms $\nu(x_i)$). The same holds for $DEC_{\ell,\ell'}$ when working with affine programs: in such a case, for each i we have $\nu(x_i) = g + \sum_{i=1}^{n} h_i \cdot \nu(x_i)$, where $g, h_1, \ldots, h_n$ are constants given by the respective assignment command in the program. Substituting these linear combinations for $\nu(x_i)$ and re-arranging again yields a linear inequality w.r.t. all $\nu(x_i)$. Let us now consider the antecendent $\nu \in I(\ell)$ common to both formulae. Since we assumed that I is polyhedral $I(\ell)$ is an intersection of half-spaces, i.e. there is an $n \times n$ matrix C and an n-dimensional vector $\boldsymbol{d}$ such that $\nu \in I(\ell)$ iff $C \cdot (\nu(x_i))_{i=1}^n \leq \boldsymbol{d}$. To streamline the notation, we abuse it a bit and denote by $\boldsymbol{\nu}$ the vector $(\nu(x_i))_{i=1}^n$. We conclude that both the conditions $NNEG_\ell$ and $DEC_{\ell,\ell'}$ can be re-written in the following generic form:

$$GEN(\boldsymbol{a}, b, \boldsymbol{d}, C) \equiv \forall \boldsymbol{\nu} \in \mathbb{R}^n : C \cdot \boldsymbol{\nu} \leq \boldsymbol{d} \implies \boldsymbol{a} \cdot \boldsymbol{\nu} < b, \tag{3}$$

for appropriate vectors $\boldsymbol{a}, \boldsymbol{d}$, matrix C, and scalar b. If all these four objects were constants, checking the validity of $GEN(\boldsymbol{a}, b, \boldsymbol{d}, C)$ would reduce to checking whether a given convex polytope is contained in a given half-space, which in turn reduces to checking the feasibility of a system of linear equations: $GEN(\boldsymbol{a}, b, \boldsymbol{d}, C)$ holds iff there is no counterexample to the implication, i.e. if the system of linear inequalities

$$\left(\frac{C}{-\boldsymbol{a}}\right) \cdot \boldsymbol{\nu} \leq \left(\frac{\boldsymbol{d}}{-b}\right) \tag{4}$$

is infeasible over $\boldsymbol{\nu} \in \mathbb{R}^n$. However, in the ranking function synthesis scenario, only C and $\boldsymbol{d}$ are constants, since they are given entirely by the invariants, which are assumed to be pre-computed before the synthesis process. On the other hand, each component of $\boldsymbol{a}$ is of the form $a_i \cdot a_i^\ell$, where a_i is a constant given by the program and a_i^ℓ is an unknown template

coefficient; a similar decomposition can be applied to b. This could seem problematic because then the system of inequalities (4) is no longer linear: both $\boldsymbol{\nu}$ and $\boldsymbol{a}$ contain unknowns. This issue is overcome by a fundamental result in mathematical optimisation called *Farkas' lemma,* from which this RF synthesis algorithm takes its name. There are several possible versions of the lemma [MG07], from which we take the one most suitable for our purposes:

Theorem 1 (Farkas' lemma)**.** *Let $A \in \mathbb{R}^{n \times n}$ and $\boldsymbol{v} \in \mathbb{R}^n$ be arbitrary. Then* exactly one *of the following statements is true*

- *The system $A \cdot \boldsymbol{x} \leq \boldsymbol{v}$ has a solution $\boldsymbol{x} \in \mathbb{R}^b$.*
- *There exists $\boldsymbol{y} \in \mathbb{R}^n$ such that $\boldsymbol{y} \geq \mathbf{0}$, $\boldsymbol{y}^\mathsf{T} \cdot A = 0$, and $\boldsymbol{y}^\mathsf{T} \cdot \boldsymbol{v} < 0$.*

A computer scientist might imagine the Farkas' lemma to state a refutation-completeness of a certain logical system. If some $\boldsymbol{x}$ satisfies $A \cdot \boldsymbol{x} \leq \boldsymbol{v}$, then for any $\boldsymbol{y} \geq \mathbf{0}$ the linear inequality $(\boldsymbol{y}^\mathsf{T} \cdot A) \cdot \boldsymbol{x} \leq \boldsymbol{y}^\mathsf{T} \cdot \boldsymbol{v}$ is also true – this can be easily checked. Hence, non-negative linear combinations of the rows of $(A \mid \boldsymbol{v})$ can be seen as *consequences* of the original system of inequalities and the process of taking such combinations as a sound inference rule producing such consequences. In particular, if the first item in the lemma's statement is true, the second one must be false, since the inequality $0 \leq$ *something negative* is false. The crux of the Farkas' lemma is that it shows the rule to be refutation-complete: if the original system is infeasible (i.e. the first item in the lemma's statement is false), then the second item must be true, and hence we must be able to infer an obvious contradiction: an inequality $0 \leq z$ for some $z < 0$.

We now apply the Farkas' lemma to the ranking function synthesis problem, namely to the system of inequalities (4), substituting

$$A \text{ with } \left(\frac{C}{-\boldsymbol{a}}\right), \text{ and } \boldsymbol{v} \text{ with } \left(\frac{\boldsymbol{d}}{-b}\right).$$

We get that (4) is infeasible (i.e., $GEN(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{d}, C)$ is valid) iff the following system has an $n+1$-dimensional solution $\boldsymbol{y} = (\boldsymbol{\mu}, \lambda) \geq \mathbf{0}$ (where $\boldsymbol{\mu}$ is n-dimensional and λ is a scalar):

$$\boldsymbol{\mu}^\mathsf{T} \cdot C - \lambda \boldsymbol{a} = \mathbf{0} \text{ and } \boldsymbol{\mu}^\mathsf{T} \cdot \boldsymbol{d} - \lambda b < 0. \tag{5}$$

It is easy to check that in any non-negative solution to (5), λ must be non-zero, i.e. positive. (Otherwise we would have $\boldsymbol{\mu}^\mathsf{T} \cdot C = 0$ and $\boldsymbol{\mu}^\mathsf{T} \cdot \boldsymbol{d} < 0$, a contradiction.) Taking one such solution and multiplying it by $1/\lambda$ again yields a feasible solution to (5) in which $\lambda = 1$. Hence, we can factor λ out of (5) entirely, getting

$$\boldsymbol{\mu}^\mathsf{T} \cdot C - \boldsymbol{a} = \mathbf{0} \text{ and } \boldsymbol{\mu}^\mathsf{T} \cdot \boldsymbol{d} - b < 0. \tag{6}$$

The system (6) contains two types of unknowns: $\boldsymbol{\mu}$, and template variables hidden in $\boldsymbol{a}$ and b. Since $\boldsymbol{\mu}$ is only multiplied with C and $\boldsymbol{d}$, the system is linear and its feasibility can be checked by linear programming, retrieving the feasible solution if one exists.

The following list summarises all the conceptual steps of the algorithm.

1. Input an affine program $\mathcal{P}$ with n variables $x_1, \ldots, x_n$, and an invariant I for the program.

2. For each location ℓ of $\mathcal{P}$, fix a linear template $b^\ell + \sum_{i=1}^{n} a_i^\ell \cdot x_i$ for a function f with unknown coefficients $b^\ell, a_1^\ell, \ldots, a_n^\ell$.

3. For each location ℓ, construct the formula $NNEG_\ell(f)$. For each pair of locations ℓ, ℓ', where ℓ' is a one-step successor of ℓ, construct the formula $DEC_{\ell,\ell'}(f)$. Gather the constructed formulae together to get a finite set of linear-arithmetic formulae $\{GEN(\boldsymbol{a}_k, b_k, \boldsymbol{d}_k, C_k) \mid k \in K\}$, where K is a finite index set. (With $\boldsymbol{a}_k$ and b_k containing template unknowns.)

4. For each $k \in K$, introduce an n-dimensional vector of variables $\boldsymbol{\mu}_k$ and construct a system $\mathcal{L}_k$ of linear (in)equalities

$$\mathcal{L}_k\colon\ \boldsymbol{\mu}_k^\mathsf{T} \cdot C_k - \boldsymbol{a}_k = \mathbf{0} \text{ and } \boldsymbol{\mu}_k^\mathsf{T} \cdot \boldsymbol{d}_k - b_k < 0.$$

5. Gather all the constructed inequalities together to obtain a system $\mathcal{L}$:

$$\mathcal{L}\colon\ \mathcal{L}_1 \wedge \mathcal{L}_2 \wedge \ldots \wedge \mathcal{L}_{|K|}.$$

6. Using linear programming, find a feasible solution of $\mathcal{L}$ (w.r.t. the set of variables $\{\boldsymbol{\mu}_k \mid k \in K\} \cup \{a_1^\ell, \ldots, a_n^\ell, b^\ell \mid \ell \text{ is a location }\}$). If no such solution exists, report that a linear ranking function supported by I does not exist. Otherwise, let $(\widetilde{a}_1^\ell, \ldots, \widetilde{a}_n^\ell, \widetilde{b}^\ell)_\ell$ be the computed feasible solution projected to the template variables. Return a linear ranking function f such that

$$f(\ell, \nu) = \widetilde{b}^\ell + \sum_{i=1}^{n} \widetilde{a}_i^\ell \cdot \nu(x_i).$$

In the last step, we use a standard trick to handle the strict inequalities in $\mathcal{L}$'s: we introduce a fresh non-negative variable ε and replace each strict inequality $L < R$ with $L + \varepsilon \leq R$. We then seek a solution maximising the value of ε, rejecting the system as infeasible if the optimal value of ε is zero.

The algorithm is not only sound, but also *relatively complete* in the sense that if there exists a linear ranking function for $\mathcal{P}$ supported by I, the algorithm will find one such function. Moreover, the algorithm is essentially a polynomial-time reduction to linear programming. Hence, the problem whether a given program admits a linear RF supported by a given invariant is solvable in polynomial time.

Algorithmic Synthesis of Ranking Functions: Specific Algorithms The earliest use of Farkas' lemma for termination proving we were able to identify is due to Colón and Sipma, who worked first with unnested loops [CS01], and then with more general programs [CS02]. They do not ultimately reduce RF synthesis to linear programming and instead present an algorithm based on the manipulation of polyhedral cones. Linear programming was utilised by Podelski and Rybalchenko [PR04a], who prove that their approach is complete (in an absolute sense) for unnested straight-line loops. The paper [BMS05] extends the template-based approach to the scenario where the invariants are not pre-computed but synthesised during the termination-proving process. The key observation is that we can set up a linear template for invariants themselves, turning the C-matrices in GEN-formulae into matrices of unknowns. This turns (6) into a system of quadratic constraints, and [BMS05] presents two heuristic approaches to solving such systems.[5]

The aforementioned work [BMS05] also considers the synthesis of lexicographic ranking functions. Indeed, the Farkas' approach can be used to synthesise individual components of a linear LexRF. The paper [ADFG10] describes an iterative technique computing the components one by one. In each iteration, a relaxed version of the linear system $\mathcal{L}$ is constructed, aiming to find a "partial" ranking function that maximises the number of program transitions along which the function value strictly decreases (on other transitions, the function must not increase). The ranked transitions are then removed from the program, the synthesised partial RF is added as the next component of a lexicographic RF, and the process proceeds to the next iteration until no unranked transitions remain. The paper [CSZ13] considers a similar approach with an important twist inspired by the approach to synthesising Ramsey-based RFs [CPR06]. The twist consists of deploying a safety prover for termination proving: when (the relaxation of) $\mathcal{L}$ becomes infeasible, it means that there is no RF supported by the given invariant. This might be because the invariant is not precise enough. Hence, a safety prover is used to check whether we can produce an execution along which the function computed so far does not lexicographically decrease. If the safety check fails, the safety prover provides such a counterexample execution. This counterexample is used to refine the vectorial function: using an adaptation of the Farkas' approach, we compute a new component that decreases along the counterexample and thus extends the scope of the termination certificate. One interesting difference between [ADFG10] and [CSZ13] is that in the former, all components of the LexRF have to be non-negative, while in the latter, each component needs to be non-negative only on that part of the program whose termination is proved by that component. These subtle differences between definitions of LexRFs and their computational implications were studied in [BG15].

Finally, we note that termination proving via synthesis of LexRFs was implemented in several successful temporal provers, such as Terminator [CPR06], T2 [BCIKP16], AProVE [Gie+17], and others. The research surrounding termination proving and ranking

functions goes well beyond the scope of discussion warranted in this thesis; we refer the reader to the survey paper [CPR11] as a starting point for further research on the topic.

4.2 History of Ranking Supermartingales

Let us now see how can the ideas from the previous section be transferred into the probabilistic setting. In this section, we describe several previous lines of research that converged towards the notion of *ranking supermartingale,* which forms the foundation of probabilistic termination proving. The work of this thesis' author, which builds on this notion to develop several new approaches to probabilistic program analysis, is described in the next chapter.

RFs and Probabilistic Programs Recall that an RF f maps program states to some well-founded set $(W, \preceq)$ in such a way that upon each execution step, the f-value of the current state strictly decreases w.r.t. the $\preceq$ ordering. Nothing prevents us from defining RFs in exactly the same way also for probabilistic programs. This implicitly amounts to treating the probability as non-determinism: the value of the function will be required to decrease for *each* outcome of any probabilistic instruction in the program. Such a definition is definitely sound w.r.t. almost-sure termination proving: the existence of such an RF demonstrates that each program's execution terminates, and thus, the probability of terminating runs is one. However, it can be easily seen that this is a rather crude tool for a.s. termination proving, as it is easy to find programs that do terminate almost-surely but still admit (possibly infinitely many) non-terminating executions (consider a simple loop simulating an asymmetric random walk over integers with negative bias and terminating once a non-positive number is reached).

Moreover, by ignoring the quantitative aspect of the probabilistic behaviour, the classical RFs are not suitable for quantitative analysis (quantitative termination, expected runtime, tail bound analysis,…).

From RFs to (Scalar) Ranking Supermartingales A reader familiar with the basic theory of random walks will likely not be surprised (in particular in light of the above example) by how to fit RFs into the probabilistic setting in a less naive and more useful way: in essence, RFs tracks the program's current "distance" from termination, and for a probabilistic system to (almost-surely) reach a certain state, it is sufficient to show that the distance towards that state decreases *in expectation*. The following definition formalises this intuition.

Definition 2 (Ranking supermartingale)**.** *Let $\mathcal{T} = (S, \Delta)$ be a PTS and $f : S \to \mathbb{R}$ a lower-bounded function. We say that f is a* ranking supermartingale (RSM) *for a target set of states $F \subseteq S$ if for every transition $\tau = (s, \overrightarrow{\tau}) \in \Delta$ s.t. $s \notin F$ it holds that*

$$\mathbb{E}_{s' \sim \overrightarrow{\tau}}[f(s')] \leq f(s) - 1.^{6}$$

Similarly to the case of classical RFs, the choice of -1 in the definition of an RSM is rather arbitrary and can be replaced by any non-positive constant.

Mathematical Martingales The term "supermartingale" might seem puzzling to readers unfamiliar with probabilistic literature. Given that RSMs seem like a straightforward probabilistic generalisation of RFs, why not use a more down-to-earth term such as a "probabilistic ranking function?" The reason is that the RSMs we just introduced are deeply connected to so-called *(super)martingale processes* in abstract probability theory. How these processes came to be named "martingales" is an interesting (and highly speculative) story that is beyond the scope of this thesis (we refer the reader to the article [Man05] for the sometimes colourful details). We just state what the (super)martingales are: a martingale is a real-valued stochastic process whose expected change in each step is zero, even if the knowledge of past values of the process is provided; in a supermartingale, the expected change of the process is non-positive, i.e. the process tends to decrease (or stay the same) over time. Formally, the definition is as follows:

Definition 3. *Let $X_0, X_1, X_2, X_3, \ldots$ be a stochastic process (a sequence of random variables in the same probability space). The process is a* martingale *if for every $i \geq 0$ it holds*

$$\mathbb{E}[X_{i+1} \mid X_0, X_1, \ldots, X_i] = X_i, \tag{7}$$

where $\mathbb{E}[\,\cdot \mid \cdot\,]$ denotes conditional expectation. If (7) holds with $\leq$ instead of $=$, we say that the process is a supermartingale; *and if it holds with $\geq$ instead of $=$, we say that the process is a* submartingale.

We note the sometimes confusing terminology: supermartingales tend to *decrease* (or stay the same) while the converse holds for submartingales. A way to imagine this is that a supermartingale is "pushed" from above, producing a downward trend.

The above definition swept a lot of technical details under the rug. In particular, defining the conditional expectation is a rather non-trivial process. We just note that the conditional expectation in (7) is a random variable, which, given a sample $\omega \in \Omega$, returns the expected value of X_{i+1} conditional on the event $X_0 = X_0(\omega) \wedge \ldots \wedge X_i = X_i(\omega)$. We also leave aside the fact that probabilistic literature (including the literature on termination proving with martingales) uses a more general definition of a martingale, which involves *filtrations* and conditioning by *sub-algebras.* These formal foundations, whose lack should not impede the comprehension of this thesis, can be found in most of the textbooks on formal probability theory (e.g. [Wil91; Bil95; Ros06]).

Example 1. *Consider the following stochastic process $(Y_i)_{i=0}^{\infty}$: we put $Y_0 = 0$. Then, we* once *flip a fair coin. If it lands on heads, in* every *subsequent time step we* increase *the value of the process by 1; if the coin lands on tails, in every subsequent time step we* decrease *the value of the process. The process $(Y_i)_{i=0}^{\infty}$ can exhibit exactly two trajectories, each with probability $\frac{1}{2}$: one trajectory is $0, 1, 2, 3, \ldots$, while the other one is $0, -1, -2, -3, \ldots$. It is easy to see that $\mathbb{E}[Y_i] = 0$*

for every $i \geq 0$. However, to process is neither a martingale nor a super- or submartingale. Indeed, once the value of Y_1 is revealed, we know for sure that the process will keep either increasing (if $Y_1 = 1$), in which case it cannot be a (super)martingale, or decreasing, in which case it cannot be a submartingale. However, if we modify the process by tossing a fair coin in every step, with the outcome of the toss deciding if the next step *should be an increment or a decrement (i.e., if we change the process into the standard symmetric random walk), then we get a martingale.*

Example 2. *Now consider the same example as before, but with an unfair coin which lands on tails with probability $\frac{2}{3}$. That is, the process $(Y_i)_{i=0}^{\infty}$ can again exhibit only two trajectories, with the decreasing one being "heavier". Compare this to the asymmetric, 0-initiated random walk $(X_i)_{i=0}^{\infty}$ where in each step there is a probability $\frac{2}{3}$ of decrementing and $\frac{1}{3}$ of incrementing. Then for all $i \geq 0$ we have $\mathbb{E}[X_i] = \mathbb{E}[Y_i] = \frac{-i}{3}$. However, for the same reasons as above, the random walk $(X_i)_{i=0}^{\infty}$ is a supermartingale, while $(Y_i)_{i=0}^{\infty}$ is not. In a sense, while it would be tempting to say that both processes have a negative downward trend, such a characterisation would be misleading for the latter one, as we can easily find ourselves in a situation where the converse is true.*

The previous examples give two well-known examples of (super)martingale processes: 1D symmetric and asymmetric random walks. They also illustrate why the martingales are relevant to proving termination properties. Indeed, the random walks in the two examples do eventually hit negative numbers with probability 1; whereas the processes $(Y_i)_{i=0}^{\infty}$ do not. In a sense, the conditional expectation constraints (7) ensure that there is a solid trend towards some set of states, unlike the weaker sense of trend pictured by the (unconditional) expected values.

The connection between ranking supermartingales (Definition 2) and mathematical martingales (Definition 3) is rather straightforward. Let $\mathcal{T}$ be a PTS and f a ranking supermartingale for a target set F. We can define a stochastic process $(M_i)_{i=0}^{\infty}$ such that $M_i = f(s_i)$, where s_i is a random variable representing the i-th state along a run in $\mathcal{T}$. It is not difficult to see that $(M_i)_{i=0}^{\infty}$ is then a supermartingale (in the mathematical sense) with two additional properties:

- it is uniformly bounded from below; and
- it satisfies a stricter decrease property than "standard" supermartingales: namely, for each i it holds that $\mathbb{E}[M_{i+1} \mid M_0, M_1, \ldots, M_i] \leq M_i - \mathbb{I}_{s_i \notin F}$, where $\mathbb{I}_E$ is the indicator function of event E.

This connection allows us to utilise the vast arsenal of tools from martingale theory in probabilistic program analysis and other areas of probabilistic verification.

Martingales in Probabilistic Verification Before we describe the usage of RSMs in termination proving, we briefly discuss the importance of martingale theory for probabilistic verification in general. The usefulness of martingales stems from the fact that many interesting theorems were proved about them, often theorems similar in spirit to those

holding for sequences of independent and identically distributed (i.i.d.) random variables. Examples of these are the *martingale central limit theorem*, which does not need any i.i.d. assumptions, or the *martingale convergence theorem,* which posits that a bounded (in a certain sense) martingale must almost-surely stabilise. Another useful tool are the *concentration inequalities* (such as Azuma's inequality) for martingales, analogues of Chernoff bounds, that work for martingales as opposed to i.i.d. variables. The fact that the i.i.d. assumption is not needed to apply these theorems is important in probabilistic verification where we work with state-based systems. The behaviour of such system in some time step depends on the current state, and the current state depends on choices made in the past, a situation not matching the i.i.d. property.[7] On the other hand, to apply martingale-based tools, we "only" need to find a martingale process connected to the probabilistic system we want to analyse (such as when we find an RSM for a given program). There is no guarantee that finding such a martingale is easy, but it is typically possible as long as the system exhibits a robust trend towards some behaviour.

We conclude this part by noting that the mathematical arsenal of martingale theory is not always necessary in order to prove something about, say, a ranking supermartingale. For instance, the fact that an RSM for the set of terminal states acts as a certificate for almost-sure termination (which the reader by now likely, and reasonably, expects to hold and which will be formally stated in the next part) can be proved by relatively short probabilistic computation, without resorting to black-box results. Still, it is useful to have the arsenal at our disposal, in particular once we start investigating more complex properties than a.s. termination.

RSMs as Certificates of Finite Termination The usage of RSMs for proving a.s. termination is not new to the author's research. In this part, we will recapitulate the history of the concept and of its applications.

We first state the, by now anticipated, correctness of RSMs.

Theorem 2. *Let $\mathcal{T} = (S, \Delta)$ be a PTS and $In, F \subseteq S$ sets of initial and terminal states. Assume that there is a RSM f for F in $\mathcal{T}_{In}$, i.e. in the restriction of $\mathcal{T}$ to states reachable from In. Then:*

a) For every $s_0 \in In$ and every scheduler σ it holds that

$$\mathbb{P}^{\sigma}_{s_0}[\textit{a state from } F \textit{ is reached}] = 1.$$

b) Denoting $Time_F$ the number of steps to reach F, it holds that

$$\sup_{\sigma} \mathbb{E}^{\sigma}_{s_0}[Time_F] \leq f(s_0).$$

In particular, the expected termination time is finite.

The proof of Theorem 2 is not difficult for someone with a fair command of formal probability theory. Still, it is an order of magnitude more intricate than the obvious

correctness of scalar RFs, which follows directly from the definition of a well-founded set. This is a pattern common to multiple methods in probabilistic program analysis: on the surface, they look (syntactically) similar to their non-probabilistic counterparts. However, proving that the methods work is typically much more intricate in the probabilistic setting and, as we shall see, one needs to be careful of subtle yet crucial details.

RSMs for Finite Termination: A History Although we presented RSMs as generalisations of ranking functions, the historical developments of the two notions seem to be independent. The earliest appearance of a concept similar to RSMs we were able to identify is due to Foster [Fos53], who studied the ergodicity of and expected return times in countable-state Markov chains with applications to queuing processes. The presented proof certificates of ergodicity eventually came to be known as *Foster-Lyapunov* functions [MT09], in a nod to Lyapunov functions from non-stochastic control theory. Foster's paper essentially contains (in a somewhat different form) the statement of Theorem 2 for countable Markov chains (i.e. countable state-space PTSs with no non-determinism). The work of Bournez and Garnier [BG05] connected the notion of Lyapunov functions to almost-sure termination of certain probabilistic rewrite systems. They also coined the term *positive termination* for almost-sure termination in a finite expected number of steps.

The notion *ranking supermartingale* was, to our best knowledge, first used in the work of Chakarov and Sankaranarayanan [CS13]. They defined RSMs for imperative arithmetic programs in a way identical in spirit to what is presented in this thesis. One of their key contributions is an algorithm for the synthesis of linear RSMs supported by a given invariant. [8] The algorithm is a straightforward syntactic extension of the procedure for the synthesis of linear scalar ranking functions, presented in Section 4.1. The crucial observation is that as long as we restricted to linear arithmetic, due to the linearity of expected value it is still possible to express the defining properties of an RSM as a conjuction of the formulae of the general form (3), i.e. $\forall \boldsymbol{\nu} \in \mathbb{R}^n : C \cdot \boldsymbol{\nu} \leq \boldsymbol{d} \implies \boldsymbol{a} \cdot \boldsymbol{\nu} < b$ (for appropriate vectors $\boldsymbol{a}, \boldsymbol{d}$, matrix C and scalar b), which can then be transformed into a conjunction of linear inequalities using the Farkas' lemma. More precisely, recall that the defining conditions of a ranking function were encoded via the formulae *NNEG* (1) and *DEC* (2). Non-negativity (or, in general, lower-boundedness) of an RSM is a property independent of any probabilities, and hence the same formula *NNEG* can be used for RSMs. To capture the expected decrease property of RSMs, we can define the following *P-DEC* formula:

$$\textit{P-DEC}_{\ell,\ell'}(f) \equiv \forall \nu : \nu \in I(\ell) \implies \mathbb{E}[f(\ell', \nu')] < b^\ell + \sum_{i=1}^{n} a_i^\ell \cdot \nu(x_i) - 1. \tag{8}$$

(Recall that a linear RF/RSM f can be represented by a set of linear coefficients $a_1^\ell, \ldots, a_n^\ell, b^\ell$). If the command effecting the transition from ℓ to ℓ' is non-probabilistic, the distribution

over the successor valuations is Dirac and (8) reduces to (2). If the command is of the form $x_i :=$ **sample**(D) for some distribution D, then

$$\mathbb{E}[f(\ell', \nu')] = b^{\ell'} + a_i^{\ell'} \cdot \mathbb{E}[D] + \sum_{\substack{j=1 \\ j \neq i}}^{n} a_j^{\ell'} \nu(x_j),$$

(where $\mathbb{E}[D]$ is the expected value of D), in which case (8) fits the general form (3). For the **if prob**(p) probabilistic branching command, the formula *P-DEC* is only parameterised by the source location ℓ, since ℓ' is a random variable. If ℓ' and ℓ'' are the possible destinations of the probabilistic **if** and **else** branches, respectively, then

$$\mathbb{E}[f(\ell', \nu')] = p \cdot (b^{\ell'} + \sum_{i=1}^{n} a_i^{\ell'} \cdot \nu(x_i)) + (1-p) \cdot (b^{\ell''} + \sum_{i=1}^{n} a_i^{\ell''} \cdot \nu(x_i)),$$

again a linear expression (as long as p is a constant hard-coded into the program) allowing re-arrangement of (8) to fit the general form (3).

Chakarov and Sankaranarayanan not only utilised the above insights to get an algorithm for RSM synthesis, but also sketched the use of martingales for proving of quantitative safety properties via the Azuma's ineuqality, a useful tool from martingale theory that we will keep encountering later. However, no automation of such safety proofs is discussed. The fundamental limitation of their work is that they only consider programs without non-determinism, i.e. Markov chains. The work of Fioriti and Hermanns [FH15] aimed to overcome this limitation and lift RSM reasoning to programs with non-determinism. Their paper provides numerous ideas relevant for the subsequent research on the topic; however, its impact was somewhat affected by its use of rather non-standard semantics. Instead of treating the program as a (possibly general state-space) MDP, they consider a construction of a single probability space which captures the behaviour of all schedulers. The main motivation for such a construction seems to be a controversial claim that under the standard semantics, RSMs are not a complete proof certificate for proving finite termination. In other words, the paper claims that there are probabilistic programs with bounded non-determinism that terminate, under every scheduler, in a finite expected number of steps, but do not admit an RSM. However, the presented counterexample does not seem to demonstrate this eventuality. The accompanying discussion focuses on the optimality of schedulers w.r.t. maximising the value of a certain variable, not on termination per se; it is not clear why should the worst-case expected termination time not work as an RSM for the program. Moreover, the arXiv paper [CF17] claims a counterexample showing that under classical semantics, the counterexample *does* admit an RSM, though the program used in [CF17] has a different syntax than the one in [FH15] (a key variable controlling a conditional branch has a Bernoulli distribution rather than a uniform one used in the original). Nevertheless, with the caveat of non-standard semantics, [FH15] does indeed prove Theorem 2 for programs with non-determinism. The work does not directly

consider automation of RSM synthesis. Instead, it designs a proof system for *compositional termination proving*, where the a.s. termination of a whole program is inferred from a.s. termination of its sub-components. Proofs derived in such a system can be viewed as (sort of) lexicographic termination arguments: by the design of the system, RSMs certifying termination of outer loops must not increase in expectation during an execution of the inner loop. While the proof system is very intuitive, Fioriti and Hermanns show that it is *not sound* in general. The issue boils down to the probabilistic notion of *uniform integrability*. Intuitively, if the RSM-defining formula involves variables that are not uniformly integrable (UI), the RSM might, with non-zero probability, keep increasing in super-polynomial pace while its expected value remains bounded. This presents an issue for the compositional proof systems, and hence non-UI variables must be excluded from RSMs. Unfortunately, as noted already in [FH15], the task of deciding whether a variable is UI is hihgly non-trivial, since programs might produce very complex distributions over their variables. Instead, the paper provides a type system through which UI of some of the program variables can be certified. It is then claimed that a compositional termination proof in which the constituent RSMs only use UI-typed variables is sound for a.s. termination. This claim again proved controversial: [HFCG19] contends (to the author's best knowledge, successfully) that the system of [FH15] is not sound under the standard semantics, likely since it does not require the constituent RSM to be integrable at the end of the while-loop it ranks. The counterexample does not involve any non-determinism, and hence the issue likely cannot be attributed to the non-standard semantics used in [FH15]. We suggest taking the aforementioned discrepancies as a reminder that probabilistic program analysis involves subtle intricacies already at the most fundamental level.

Martingales in Other Fields of Computer Science. We conclude this survey part by noting that probabilistic program analysis is by far not the only field of computer science where martingales are useful. (A fact that should not be surprising given the previously discussed usefulness of martingales in probabilistic verification.) Martingales play an important role in stochastic control theory. Already the classical book of Kushner [Kus71] details the use of martingales (and in particular, of the Doob's martingale inequality) in stability analysis of stochastic processes. This theme has been further developed in a more recent work, in which notions such as level sets [TA11] and barrier certificates [PJP04] that can be interpreted in the language of martingale theory, are used to prove various safety and liveness properties of stochastic dynamical systems. While the focus in control theory is more on the usage of such certificates rather than their automated synthesis, recent approaches consider automation via reduction to sums-of-squares programming [ST12].

A last but important category of work we mention here is the one on verification of one- and multi-counter Markov decision processes and games [BBEKW10; BBE10; BBEK11], including the work co-authored by the author of this thesis [BKNW12; BKKNK14; BKKN15]. These models can be imagined as finite-state Markov chains/Markov decision processes/games equipped with one or more integer-valued counters, which can in turn be viewed

as a very restricted class of probabilistic programs. This restricted form ensures that the models *always* possess a martingale of a nice syntactic shape that can be used to analyse (and decide) most of the verification questions we study in this chapter. While the algorithmic aspects of these works are not readily transferable to general probabilistic programs, the martingale-based arguments therein were important stepping stones towards the more general techniques presented in the next chapter.

CODE SAFETY: AUTHOR'S CONTRIBUTION

5

The previous chapter summarised state of the art in martingale methods for probabilistic program analysis at the time when the author of this thesis started contributing to this field. In this section, we present five papers (and one book chapter) from the period 2016–2021 which encompass the current state of this contribution.

5.1 Scalar RSMs: Algorithms, Complexity, Tail Bounds

The first paper we cover is the POPL'16 paper co-authored with Chatterjee, Fu, and Hasheminezhad, together with its journal version published in TOPLAS:

- K. Chatterjee, H. Fu, P. Novotný, and R. Hasheminezhad. "Algorithmic analysis of qualitative and quantitative termination problems for affine probabilistic programs". In: *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*. Ed. by R. Bodík and R. Majumdar. ACM, 2016, pp. 327–342. DOI: 10.1145/2837614.2837639. URL: http://doi.acm.org/10.1145/2837614.2837639
- K. Chatterjee, H. Fu, P. Novotný, and R. Hasheminezhad. "Algorithmic Analysis of Qualitative and Quantitative Termination Problems for Affine Probabilistic Programs". In: *ACM Trans. Program. Lang. Syst.* 40.2 (2018), 7:1–7:45. DOI: 10.1145/3174800. URL: https://doi.org/10.1145/3174800

The paper extends both the work of Chakarov and Sankaranarayanan [CS13] and of Fioriti and Hermanns [FH15] by considering RMSs for programs with non-determinism *under the standard MDP semantics*. Furthermore, we consider programs with both angelic and demonic non-determinism, the algorithmic and complexity aspects of RSM synthesis, and the derivation of tail bounds on expected termination time.

More precisely, the first contribution of the paper was the proof of Theorem 2 for affine probabilistic programs with demonic or angelic non-determinism, or even a combination thereof. Note that the formulation of Theorem 2 in this thesis considers only demonic non-determinism. Incorporating also angelic non-determinism entails changing the statement of item a) into *"For every $s_0 \in In$ there exists and angelic scheduler σ_A s.t. for all demonic schedulers σ_D it holds that…(the program terminates a.s. under σ_A and σ_D)."* Similarly, item b) changes into $\inf_{\sigma_A} \sup_{\sigma_D} \mathbb{E}_{s_0}^{\sigma_A,\sigma_D}[Time_F] \leq f(s_0)$. Next, we provided a careful definition of a linear RSM and of a Farkas'-based algorithm for its synthesis. While the formulation

of the algorithm is a rather straightforward extension of the ideas of [CS13] and of the previous work on non-probabilistic ranking functions, we were, to our best knowledge, the first to provide such a detailed recipe on how to produce the individual constraints, along with the proof of their correctness. (Indeed, to our best knowledge, none of the previous work provided a complete description of the Farkas' translation process, even to the level of detail provided in Section 4.1).

An entirely novel aspect of our formulation is the handling of angelic non-determinism, for which the Farkas' transformation yields a system of quadratic constraints. Intuitively, this is because for angelic non-determinism, the formula (3) can have a disjunction of multiple linear inequalities on its right-hand side. Indeed, in the angelic case we want to express the property *"for each state in the invariant, at least one of the available transitions decreases the value of the RSM"*, which yields a formula of the form

$$C \cdot \boldsymbol{\nu} \leq \boldsymbol{d} \Longrightarrow \bigvee_{j=1}^{m} \boldsymbol{a}_j \cdot \boldsymbol{\nu} < b_j,$$

where j ranges over choices available to the angelic scheduler. The validity of the formula is equivalent to the unsatisfiability of the formula

$$C \cdot \boldsymbol{\nu} \leq \boldsymbol{d} \wedge \bigwedge_{j=1}^{m} -\boldsymbol{a}_j \cdot \boldsymbol{\nu} \leq -b_j,$$

which, using the Farkas' transformation, yields a variant of (5) of the following form:

$$\boldsymbol{\mu}^{\mathsf{T}} \cdot C - \sum_{j=1}^{m} \lambda_j \boldsymbol{a}_j = 0 \text{ and } \boldsymbol{\mu}^{\mathsf{T}} \cdot \boldsymbol{d} - \sum_{j=1}^{m} \lambda_j b_j < 0. \tag{9}$$

The presence of multiple λ-variables makes it impossible to factor them out easily, and since $\boldsymbol{a}_j$ contains RSM template variables, (9) is a quadratic system. Of course, if the analysed program does not involve angelic non-determinism, the produced constraints are just linear.

Another contribution of the paper is the study of the complexity of RSM synthesis. More precisely, we consider the following decision problem: given an affine program $\mathcal{P}$ and its polyhedral invariant I, decide if there exists a linear scalar RSM for $\mathcal{P}$ supported by I. For programs without angelic non-determinism, the Farkas' transformation reduces the problem to checking the feasibility of a system of linear inequalities, demonstrating its solvability in polynomial time. For programs with angelic non-determinism, we present a reduction from 3-SAT showing that the problem is NP-hard. This demonstrates that, unless P = NP, the quadratic system (9) cannot be replaced by a linear one.

Finally, we consider problems pertaining to expected termination time. Theorem 2 gives an upper bound on the expected termination time, though there is no guarantee on how tight the bound is. For a class of programs with *bounded updates*, where there is a bound

δ on the one-step change of any program variable, we can obtain further information about the distribution of termination time via *martingale concentration inequalities,* such as Azuma's inequality:

Theorem 3 (Azuma)**.** *Let $(X_i)_{i=0}^{\infty}$ be a supermartingale and let there be $c > 0$ s.t. $|X_{i+1} - X_i| \leq c$ for any $i \geq 0$. Then, for any $n \geq 0$ and any $t > 0$ it holds*

$$\mathbb{P}[X_n - X_0 \geq t] \leq \exp\left(-\frac{t^2}{2nc^2}\right).$$

The Azuma's inequality can be employed (for demonic non-determinism) as follows. Let f be an RSM for our program, and suppose that we are able to derive some bound δ on the one-step changes of f. By definition of an RSM, the expected value of f decreases by at least one in every step (unless the process has already terminated). This decrease can be compensated by adding unity in every step without damaging the supermartingale property. In other words, the process $X_n = f(s_n) + \min\{n, \mathit{Time}\}$ (where s_n is the state at time n and *Time* is the termination time) is a supermartingale in the mathematical sense, and its one-step change is bounded by $c = \delta + 1$. We then observe that for any $t \geq 0$, the event $\mathit{Time} > n$ is contained in the event $X_n - X_0 \geq n - f(s_0)$. This is because

$$\begin{aligned} X_n - X_0 &= f(s_n) + n - f(s_0) && \text{(since } T > n\text{)} \\ &\geq n - f(s_0) && \text{(since } f \text{ is non-negative in non-terminal states)} \end{aligned}$$

In particular $\mathbb{P}^{\sigma}_{s_0}[\mathit{Time} \geq n] \leq \mathbb{P}^{\sigma}_{s_0}[X_n - X_0 \geq n - f(s_0)]$ under any scheduler σ. To bound the latter quantity, we observe that for large enough n, we have $n - f(s_0)$ positive. Hence, we can plug the values $n, c = \delta + 1$, and $t = n - f(s_0)$ into Azuma's inequality to get

$$\mathbb{P}^{\sigma}_{s_0}[\mathit{Time} \geq n] \leq \exp\left(-\frac{(n - f(s_0))^2}{2nc^2}\right) \leq H \cdot \exp\left(-\frac{n}{2c^2}\right),$$

where H is a constant independent of n. Thus, we have obtained an exponentially decreasing tail bound on the termination time. The argument for programs with angelic non-determinism is essentially the same; we just need to fix an angelic scheduler which decreases f in expectation. From an algorithmic perspective, the only issue to resolve is obtaining the bound δ. But the requirement $-\delta \leq f(s) - f(s') \leq \delta$ (where s' is a successor of s) can be encoded directly into the linear system through which we seek f, in a similar way as the non-negativity and decrease conditions. (In the constraints, δ is treated as a variable.) Naturally, not every program admits an RSM with bounded differences, but if a program admits a bounded-difference RSM supported by a given invariant, the aforementioned method is guaranteed to find it.

The exponentially decreasing tail bound, once obtained, can be (at least in theory) used to approximate the expected termination time up to arbitrary precision, provided that all probability distributions used in the program have a finite support. The idea is to unfold the program into a tree of computations up to a depth d computed in such a way that the

probability of not terminating before time d is extremely low and thus behaviour past step d influences the expected termination time only in a marginal way. Note that computing such d is possible given the tail bound, since the probability of non-termination by time n decreases exponentially fast, while the termination time increases only linearly fast (by 1) with increasing n. The termination time within the tree can then be computed exactly by standard techniques for Markov chains. This method is likely not too practical, since the required depth d can be exponentially large, and the whole unfolding can thus have a doubly-exponential size. We supplement this observation with a theoretical hardness result. For any constant K we define the following decision problem $\mathit{TimeDec}(K)$: given a program $\mathcal{P}$ and a number N such that the termination time of $\mathcal{P}$ is either $\leq N$ or $\geq K \cdot N$, decide which of the two cases holds. We show that for any K, $\mathit{TimeDec}(K)$ is PSPACE-hard, via reduction from the membership problem for linearly bounded Turing machines. In other words, already approximating the termination time up to an arbitrary factor is difficult. The hardness result holds even if we restrict to programs $\mathcal{P}$ that are non-probabilistic, without non-determinism, with bounded variable updates and admitting a linear RSM with bounded differences.

5.2 Repulsing Supermartingales: Quantitative Reachability, Safety, and More

Next, we cover the following POPL'17 paper co-authored with Chatterjee and Žikelić:

- K. Chatterjee, P. Novotný, and D. Žikelić. "Stochastic invariants for probabilistic termination". In: *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL 2017)*. Ed. by G. Castagna and A. D. Gordon. ACM, 2017, pp. 145–160. DOI: 10.1145/3009837. URL: http://dl.acm.org/citation.cfm?id=3009873

The initial motivation of the paper was to study quantitative termination, i.e. computing good lower bounds on termination probability in programs with demonic nondeterminism. We proposed a method of *stochastic invariants* which reduces the problem to safety analysis, and introduce the concept of *repulsing supermartingales* as a sound proof rule for quantitative safety properties.

Our paper proposes solving the aforementioned problem using the following notion of *stochastic invariants:*

Definition 4. *Let* (SI, p) *be a tuple such that* SI *is a function mapping each program location to a set of variable valuations and* $p \in [0, 1]$ *a probability. The tuple* (SI, p) *is a* stochastic invariant *if, under every scheduler, the probability of reaching a state* (ℓ, ν) *s.t.* $\nu \notin SI(\ell)$ *is at most* $1 - p$.

The name *stochastic invariant* captures the idea that while SI might not be an invariant of the program, since its complement can be reached, it "behaves like an invariant with probability at least p."

The relationship between stochastic invariants and lower bounds on termination probability is captured in the following theorem.

Theorem 4. *Let $\mathcal{P}$ be a probabilistic program with a set of terminal states F and (SI, p) its stochastic invariant. Assume that there is a mapping f from program states to real numbers satisfying the following conditions:*

- *f is bounded from below; and*
- *for every transition $\tau = (s, \overrightarrow{\tau}) \in \Delta$ s.t. $s \in SI$ and $s \notin F$ it holds that*

$$\mathbb{E}_{s' \sim \overrightarrow{\tau}}[f(s')] \leq f(s) - 1.$$

Then, under any scheduler, $\mathcal{P}$ terminates with probability at least p.

The conditions on f in Theorem 4 essentially requires f to be an RSM in a PTS obtained from the original program by restricting the set of states to SI. In particular, if $\mathcal{P}$ is affine and SI is given as a mapping from program locations to convex polytopes, then the existence of a linear f from Theorem 4 can be encoded into a set of linear constraints using the Farkas' transformation, with a program invariant I replaced by the stochastic invariant SI.

The above result shows that stochastic invariants can be used to obtain lower bounds on termination probability. The question is how to obtain such stochastic invariants. In the paper, we first focus on the following problem: given an affine program $\mathcal{P}$, its (classical) invariant I, and a polyhedral mapping SI, compute (a non-trivial value of) p such that (SI, p) is a stochastic invariant. This is essentially a quantitative safety verification problem: we want to find an upper bound p on the probability of reaching the complement of SI. In the previous work, various automated approaches to quantitative safety were considered, based, e.g. in optimised simulations [Sam+14], symbolic execution [SCG13], or finite-state abstractions [HWZ08; KKNP09]. In our paper, we stayed true to the focus on martingale-based techniques and proposed proving of quantitative safety bounds via a new notion of *repulsing supermartingales:*

Definition 5. *Let $\mathcal{P}$ be a probabilistic program with a state set S, I its (classical) invariant, and V a set of violating states. A map $f : S \to \mathbb{R}$ is an* repulsing supermartingale (RepSM) *for V supported by I if it satisfies the following:*

- *for every $s \in I \cap V$, $f(s) \geq 0$;*
- *for each initial state s_0 it holds $f(s_0) < 0$; and*
- *for every $s \in I \cap (S \setminus V)$, $\mathbb{E}_{s' \sim \overrightarrow{\tau}}[f(s')] \leq f(s) - 1$.*

Hence, for a RepSM f, we can view $-f(s)$ as a "distance" of a state s from the "boundary" of the sets of violating and non-violating states. The definition of a RepSM requires that as long as we stay in the non-violating region, the distance tends to increase over time, and hence the program tends to stay safe. This intuition can be formalised to get guaranteed bounds on the probability of reaching V.

Theorem 5. *Let $\mathcal{P}$ be a probabilistic program, $c > 0$, V a set of violating states, and let f be a RepSM for $\mathcal{P}$ (supported by some invariant I) such that f has c-bounded differences; that is, $|f(s) = f(s')| \leq c$ for each state-successor pair $s, s' \in I$. Then, under every scheduler sigma and for each initial state s_0 it holds:*

$$\mathbb{P}^{\sigma}_{s_0}[\textit{a state of } V \textit{ is reached}] \leq C \cdot \frac{\gamma^{\lceil \frac{|f(s_0)|}{c} \rceil}}{1-\gamma},$$

where $\gamma = \exp\left(-\frac{1}{2(c+1)^2}\right)$ and $C = \exp\left(-\frac{|f(s_0)|}{(c+1)^2}\right)$.

Note that the bound in Theorem 5 is exponentially decreasing in the "distance" of s_0 from V. The theorem follows by applying the Azuma's inequality to the supermartingale $X_n = f(s_n) + n$ (stopped at the first point in time in which V is reached). It is sufficient to observe that being inside V in step n entails $X_n - X_0 \geq n - f(s_0)$, and by Azuma's inequality, the probability of this event is bounded by $\exp(-(n - f(s_0))^2/2n(c+1)^2)$. Summing these bounds (as a geometric series) over all $n \geq \frac{\lceil |f(s_0)| \rceil}{c}$ (a lower bound on the number of steps needed to hit V) and simplifying yields the bound from the theorem.

We note that this use of martingales for quantitative safety proving bears some similarity to the technique of *barrier certificates* in stochastic control theory [PJP04]. The latter are mostly based on the Doob's martingale inequality, which does not require the c-bounded difference assumption at the expense of providing only linearly decreasing bounds. Our derivation of exponentially decreasing bounds is inspired by the analysis of one-counter systems [BKK14].

We now turn to algorithmic aspects of stochastic invariants and repulsing supermartingales, for which we again restrict to affine probabilistic programs. Note that the definition of a RepSM is, from a high-level point of view, syntactically similar to the definition of an RSM: a list of requirements, each essentially of the form "for each state in a given invariant, some affine relationship must hold between the current value of the function and the expected next-step value." Hence, given the set V and the invariant I, the conditions in Definition 5 can be translated into a system of linear inequalities using the Farkas' lemma. Similarly, once we have identified some stochastic invariant (SI, p), checking whether it supports an RSM (and thus getting a lower bound on the probability of termination) reduces quite straightforwardly to solving a linear system. A more interesting algorithmic question is computing the termination certificate and the stochastic invariant *at the same time:* that is, given a classical invariant I, we want to compute:

- a map SI mapping locations to convex polytopes;

- a RepSM f for the set $V = S \setminus \bigcup_\ell (\{\ell\} \times SI(\ell))$ supported by I; and
- an RSM supported by $SI \vee I$.

Using f, we can derive (via Theorem 5) a bound p s.t. (SI, p) is a stochastic invariant, and the RSM then certifies that the program terminates with probability at least p. To compute all the required objects simultaneously, we can generalise the known technique for computing ranking functions *together* with their supporting invariants by means of quadratic programming [CSS03]. The idea is that if the invariant I is unknown, then also its corresponding matrix C in (3) is composed of unknown coefficients. (Of course, we need to know at least the dimension of C so as to set up the system (3), which we can achieve by fixing in advance the number of half-spaces that define the invariant in each location.) After the Farkas' transformation, we get the system (6) in which C is multiplied with the vector of dual variables $\boldsymbol{\mu}$, thus yielding a quadratic system. The idea generalises rather straightforwardly to computing RSM, RepSM, and a stochastic invariant at once. While non-linear constraint solving is algorithmically much harder than linear one, for smaller programs, the approach is expected to be within the capabilities of modern solvers.

Our paper details several other uses of RepSMs, the chief of which is the refutation of a.s. and finite-time termination (in the weak sense that the refuted program does not, say, terminate a.s. under *any* scheduler). Let us first introduce a weaker notion of a RepSM:

Definition 6. *Let $\mathcal{P}$ be a probabilistic program with a state set S, I its (classical) invariant, and V a set of violating states. A map $f : S \to \mathbb{R}$ is a* weakly repulsing supermartingale (WRepSM) *for V supported by I if it satisfies the first two items from the definition of a RepSM (Definition 5) and moreover, for every $s \in I \cap (S \setminus V)$ it holds that $\mathbb{E}_{s' \sim \overrightarrow{\tau}}[f(s')] \leq f(s)$.*

Then, we can prove, using the *optional stopping theorem* from martingale theory, that bounded-difference RepSMs for the set of terminal states refute (existential) a.s. termination, while bounded-difference WRepSMs refute (existential) finite-time termination:

Theorem 6. *Let V be the set of terminal states of a program $\mathcal{P}$.*

1. *If there exists $c > 0$ and a RepSM for V with c-bounded differences supported by some invariant, then $\mathcal{P}$ terminates with probability strictly less than one under every scheduler.*
2. *If there exists $c > 0$ and a WRepSM for V with c-bounded differences supported by some invariant, then the expected termination time of $\mathcal{P}$ is infinite under every scheduler.*

5.3 Lexicographic Ranking Supermartingales

The previous papers on use of martingales in probabilistic program analysis considered only scalar martingale-based certificates. In the following paper, co-authored by Agrawal and Chatterjee, we were the first to introduce the concept of *lexicographic* ranking supermartingales.

- S. Agrawal, K. Chatterjee, and P. Novotný. "Lexicographic ranking supermartingales: an efficient approach to termination of probabilistic programs". In: *Proc. ACM Program. Lang.* 2. (Proceedings of POPL'18) (2018), 34:1–34:32. DOI: 10.1145/3158122. URL: https://doi.org/10.1145/3158122

The motivation for using vectorial certificates instead of scalar ones was already discussed for the non-probabilistic case. However, the process of extending lexicographic RFs to the probabilistic setting is quite intricate. Consider, for instance, one of the possible definitions of a LexRF, which [BG15] call *Bradley-Manna-Sipma (BMS)* LexRFs:

Definition 7. *Let $f : S \to \mathbb{R}^n$ be a function and denote by f_i the projection of f to the i-th component. We say that f is a* BMS-LexRF *supported by an invariant $I \subseteq S$ if for every reachable non-terminal state $s \in I$ and its successor $s' \in I$ there exists $1 \leq i \leq n$ such that all of the following conditions hold:*

$$f_i(s) \geq 0, \tag{10}$$

$$\forall 1 \leq j < i : \; f_j(s') \leq f_j(s), \tag{11}$$

$$f_i(s') \leq f_i(s) - 1. \tag{12}$$

The condition (10) is simply called *non-negativity*, the condition (11) is known as *unaffecting* condition, and (12) as *ranking* condition. It is not difficult to prove that if a program admits a BMS-LexRF, then it terminates. To lift the construction into the probabilistic setting, it would be tempting to follow the lead from the scalar setting and simply replace the right-hand sides in (11) and (12) with $\mathbb{E}_{s' \sim \overrightarrow{\tau}}[f_j(s')]$ and $\mathbb{E}_{s' \sim \overrightarrow{\tau}}[f_i(s')]$, respectively (requiring the conditions to hold for any transition outgoing from s). However, such a proof rule *would not be sound* for proving a.s. termination. As a counterexample, one can actually take the program from [FH15] demonstrating that the compositional approach to a.s. termination proving is not sound without uniform integrability constraints. The "level of unsoundness" is substantial: the program does not have a single terminating execution, yet it admits the naive probabilistic extension of BMS-LexRF.

The problem with Definition 7 is that the non-negativity condition (10) is *partial*: for each transition, we only require non-negativity of the leftmost component decreased by that transition. In our paper, we showed that probabilistic extensions of LexRFs are sound if we require *universal non-negativity*:

Definition 8. *Let $\mathbb{P}$ be a probabilistic program and I its invariant. We say that a function $f : S \to \mathbb{R}^n$ is a* lexicographic ranking supermartingale (LexRSM) *if for every state $s \in I$ and every transition $\tau = (s, \overrightarrow{\tau})$ the following holds:*

- *$\forall i \in \{1, \ldots, n\} : \; f_i(s) \geq 0$; and*
- *if s is non-terminal, then there exists $i \in \{1, \ldots, n\}$ such that:*

$$\forall 1 \leq j < i : \; \mathbb{E}_{s' \sim \overrightarrow{\tau}}[f_j(s')] \leq f_j(s),$$
$$\mathbb{E}_{s' \sim \overrightarrow{\tau}}[f_i(s')] \leq f_i(s) - 1.$$

We prove that LexRSMs are sound for proving a.s. termination:

Theorem 7. *Suppose that $\mathbb{P}$ admits a LexRSM f supported by some invariant I. Then $\mathbb{P}$ terminates almost-surely under any scheduler.*

Several comments on our result our in order. First, although LexRSMs are syntactically a rather straightforward extension of RFs (the universal non-negativity was previously used in the context of LexRFs in [ADFG10]), this does not hold for proving their soundness. As demonstrated by the unsoundess of partial non-negativity, the proof requires delicate reasoning about the underlying stochastic processes. As a matter of fact, in our paper we first provide an abstract definition of a LexRSM (in the language of general stochastic processes) and a proof of their soundness. Only then we instantiate them to the domain of probabilistic program analysis. Hence, our result has the potential for applicability also in other application domains.

Second, Theorem 7 does not guarantee that the program terminates in a finite expected number of steps. Indeed, we give an example of a linear-arithmetic program without non-determinism which has infinite expected termination time and yet admits a LexRSM (and terminates almost-surely). While this could be viewed as a weakness of LexRSMs, we argue that it is actually a feature, as it demonstrates that LexRSMs are a more general proof rule than scalar RSMs, which can only exist for programs that do terminate in finite expected time under every scheduler. We also formulate some additional conditions under which LexRSMs entail finite (and polynomial) bounds on the expected termination time.

Finally, we comment on the relationship with the uniform integrability issue studied in [FH15]. One might suspect whether the universal non-negativity condition simply another way of ensuring uniform integrability and whether our method does actually bring something new over the typechecking-supported compositional termination rule from [FH15]. Neither is the case. First, a sequence of non-negative random variables does not need to be uniformly integrable and vice versa. So on the mathematical level, the proof rules are orthogonal. One might wonder whether this orthogonality survives in the concrete world of affine probabilistic programs. To examine this, we designed a LexRSM-based compositional termination rule which replaces the uniform integrability requirement from [FH15] with non-negativity of the constituent RSMs (and thus dispenses with the need for typechecking). We show that this new rule is able to prove a.s. termination of programs that cannot be handled by the rule from [FH15]. We also note that while the soundness of the proof rule from [FH15] under standard MDP semantics has been questioned [HFCG19], our proof rule was proven sound under this standard semantics. (And fortunately, it so far survived all possible counterexamples: the example from [HFCG19] permits some components of the proof certificate to become unboundedly negative, and hence does not contradict the soundness of LexRSMs.)

Finally, we comment on the algorithmic aspects of LexRSM synthesis, for which we again restrict to affine programs and focus on LexRSMs with linear components. Since LexRSMs are syntactically similar to non-negative LexRFs studied in [ADFG10],

```
ℓ0 : while y ≥ 0 do
       x := y;
ℓ1 :   while x ≥ 0 do
         x := x − 1 + Norm(0, 1)
       od;
       y := y − 1
     od
```

(a)

```
ℓ0 : while x ≥ 0 do
       if y ≥ 0 then
         y := y + Uni[−7, 1]
       else
         x := x + Uni[−7, 1] ;
ℓ1 :   y := y + Uni[−7, 1]
       fi od
```

(b)

Figure 2: Motivating examples. $Norm(\mu, \sigma)$ samples from the normal distribution with mean μ and std. deviation σ. $Uni[a, b]$ samples uniformly from the interval $[a, b]$. Location labels are the "ℓ_i": one location per loop head and one additional location in (b) so as to have one assignment per transition (a technical requirement for our approach).

we can employ the iterative Farkas'-based algorithm from that paper modified to handle probabilistic programs in a similar way in which [CS13] extended algorithms for scalar RF synthesis to the probabilistic setting. The resulting algorithm runs in polynomial time and is guaranteed to find a linear LexRSM of the smallest degree supported by the input invariant I.

5.4 Beyond Non-Negative LexRSMs

As shown in the previous section, LexRSMs with relaxed non-negativity conditions are not necessarily sound certificates of a.s. termination. On the other hand, universal non-negativity might be too restrictive. Indeed, there is a spectrum of non-negativity conditions between the universal non-negativity of LexRSMs and the single-component non-negativity of Bradley-Manna-Sipma LexRFs, and there remains hope that some relaxation of universal non-negativity is possible while retaining soundness. Such relaxations were explored in the Formal Methods'21 paper co-authored with Chatterjee, Goharshady, Zárevúcky, and Žikelić:

- K. Chatterjee, E. K. Goharshady, P. Novotný, J. Zárevúcky, and D. Žikelić. "On Lexicographic Proof Rules for Probabilistic Termination". In: *Proceedings of FM'21*. Ed. by M. Huisman, C. S. Pasareanu, and N. Zhan. Vol. 13047. LNCS. Springer, 2021, pp. 619–639

As a motivating example, consider the two programs in Figure 2. The program in Figure 2a does terminate a.s., as can be shown by a simple random-walk argument. A linear LexRSM proving this needs to have a component containing a positive multiple of x at the head of the inner while-loop (ℓ_1). However, due to the sampling from the normal

distribution, which has unbounded support, the value of x inside the inner loop cannot be bounded from below. Hence, the program does not admit a linear LexRSM. In general, LexRSMs with strong non-negativity do not handle well programs with unbounded-support distributions. Now consider the program in Figure 2b. It can be again shown that it terminates a.s.; however, this cannot be witnessed by a linear LexRSM: to rank the "if-branch" transition, there must be a component with a positive multiple of y in ℓ_0. But y can become arbitrarily negative within the else-branch, and cannot be bounded from below by a linear function of x. While it is possible that both programs do admit a non-linear LexRSM, restrictions to linear arithmetic are typically preferred due to easier automation.

We first consider a probabilistic generalisation of so-called *Ben-Amram–Genaim (BG)* LexRFs [BG15]. Intuitively, a BG-LexRSM replaces the universal non-negativity (first item in Definition 8) with a weaker version requiring that if s is non-terminal and i is the leftmost component s.t. $\mathbb{E}_{s' \sim \overrightarrow{\tau}}[f_i(s')] \leq f_i(s) - 1$, then for all $1 \leq j \leq i$ we have $f_j(s) \geq 0$. Unfortunately, we show that already this relaxation is unsound. By examining the counterexample, we identify a supplementary condition which ensures soundness, yielding the notion of a *generalised LexRSM (GLexRSM):*

Definition 9. *Let $\mathbb{P}$ be a probabilistic program and I its invariant. We say that a function $f : S \to \mathbb{R}^n$ is a* generalised lexicographic ranking supermartingale (GLexRSM) *if for every non-terminal state $s \in I$ and every transition $\tau = (s, \overrightarrow{\tau})$ there exists $1 \leq i \leq n$ such that the following holds:*

$$\forall 1 \leq j < i : \mathbb{E}_{s' \sim \overrightarrow{\tau}}[f_j(s')] \leq f_j(s), \tag{13}$$

$$\mathbb{E}_{s' \sim \overrightarrow{\tau}}[f_i(s')] \leq f_i(s) - 1, \tag{14}$$

$$\forall 1 \leq j \leq i : f_j(s) \geq 0, \tag{15}$$

$$\forall 1 \leq j \leq i : \mathbb{E}_{s' \sim \overrightarrow{\tau}}[f_j(s') \cdot \mathbb{I}_{<j}(s')] \geq 0, \tag{16}$$

where $\mathbb{I}_{<j}(s')$ is the indicator function of the set of all states in which a transition ranked by some component left to the j-component is enabled.

We call the requirement (15) *partial non-negativity* and (16) *expected leftward non-negativity (ELN).* It is the ELN requirement that distinguishes GLexRSMs from a probabilistic generalisation of BG-LexRFs.

We prove the soundness of GLexRSMs for proving a.s. termination, using a novel application of the Borel-Cantelli lemma.

Theorem 8. *Suppose that $\mathbb{P}$ admits a GLexRSM f supported by some invariant I. Then $\mathbb{P}$ terminates almost-surely under any scheduler.*

In particular, GLexRSMs can prove a.s. termination of the programs in our motivating examples.

We now turn to the automation of GLexRSM synthesis. Unfortunately, even when restricted to linear-arithmetic programs, the ELN requirement (16) cannot be straightforwardly translated into a set of linear constraints, since it involves a rather complex integral. Hence, we consider adding further restrictions on the shape of the program or on the GLexRSM itself so that the integration can be replaced by simpler operations. While these restrictions are less general than GLexRSMs, they are (in a linear form) still applicable to a wider class of programs than LexRSMs.

First, we consider programs in which all samplings are from bounded-support distributions (such as the program in Figure 2b). We show that in such programs, the existence of a linear GLexRSM is equivalent to the existence of a linear function $f\colon S \to \mathbb{R}^n$ whose definition differs from a GLexRSM in that:

- (16) is only required to hold for transitions corresponding to probabilistic branching, and
- for all other transitions, f satisfies the following *expected non-negativity (EN)* property (recall that i denotes the ranking component):

$$\forall 1 \leq j \leq i : \mathbb{E}_{s' \sim \vec{\tau}}[f_j(s')] \geq 0. \tag{17}$$

The existence of such a function (supported by a given invariant) can be decided by adapting the iterative Farkas'-based algorithm for LexRSMs. For the modification, one needs to observe that (17) can be encoded into a linear constraint by the Farkas' transformation. Encoding (16) for probabilistic branching locations is a more technical step that rests on two ideas:

- Each probabilistic state has just two successors, so the integral in (16) is a two-term sum involving the indicators $\mathbb{I}_{<j}(s')$.
- By the time we are synthesising the j-component of f, we already know the previous components and thus can identify the transitions ranked by them (and thus also the valuations in which guards of these transitions are satisfied, which can be described by a linear-arithmetic formula). Hence, we can remove the indicators from the sum and instead replace the universal quantification over "all states in the invariant" by "all states in the invariant whose successors satisfy a guard of some transition ranked by a previous component". Since the latter set can be expressed as a union of convex polytopes, we get a formula that can be fed into the Farkas' transformation.

We note that this modification of GLexRSMs is still sufficient to prove the a.s. termination of the program in Figure 2b.

We then study how to drop the bounded-support assumption without sacrificing efficient automation. First, we impose a mild syntactic restriction on the programs we analyse: if there are two transitions, one corresponding to probabilistic branching, the

other to a sampling instruction, then these two transitions must not share a target location. Such a property can be ensured by using dummy *skip* statements. Second, we impose restrictions on the form of linear "GLexRSMs" that we aim to synthesise. More specifically, we say that a linear function $f : S \to \mathbb{R}^n$ satisfies the *UNBOUND* condition if the following holds:

- Let τ be any transition which changes some variable x by sampling from a distribution that has unbounded support (in particular, due to the syntax of PPs, all successor states of τ share the same location ℓ). Moreover, let i be the smallest index such that $\mathbb{E}_{s' \sim \overrightarrow{\tau}}[f_i(s')] \leq f_i(s) - 1$. Then for all $1 \leq j < i$ the coefficient of variable x in the linear function f_j at the source location of τ is zero.

We prove that a program terminates a.s. whenever it admits a linear function $f : S \to \mathbb{R}^n$ that satisfies the UNBOUND condition *and* the definition of a GLexRSM with the exception of:

- ELN property (16) being only required for probabilistic branching transitions; and
- the EN property (17) being satisfied for all the other transitions.

The existence of such a function can be again decided by a modification of the iterative algorithm for LexRSM synthesis. This modification of GLexRSMs is sufficient to prove a.s. termination of the program in Figure 2a

5.5 Summary of Martingale-Based Techniques: A Book Chapter

The results from the previous three sections, as well as several results of other authors [MMKK18], were summarised in a chapter of *Foundations of Probabilistic Programming* co-authored with Chatterjee and Fu:

- K. Chatterjee, H. Fu, and P. Novotný. "Termination Analysis of Probabilistic Programs with martingales". In: *Foundations of Probabilistic Programming*. Ed. by G. Barthe, J.-P. Katoen, and A. Silva. Cambridge University Press, 2020. Chap. 7, pp. 221–258

5.6 Non-Probabilistic Tech Transfer: Proving Non-Termination by Program Reversal

Somewhat ironically, we conclude this section with a result from *non-probabilistic* program analysis. However, there *is* a probabilistic connection: the result spun off preceding

discussions on *eventually valid* a.s. termination certificates, e.g. RSMs that only start to behave as RSMs after some finite but possibly unbounded number of steps. This resulted in discussions of *eventual invariants,* i.e. sets of states that one is guaranteed to reach eventually. Such a concept can be seen as an instance of reachability and thus also termination analysis in non-probabilistic programs, which led us to have a closer look at the relationship between invariants and termination. A final product of this study was a new approach for automated proving of *non-termination* in non-probabilistic programs, presented in the following PLDI'21 paper co-authored with Chatterjee, Goharshady, and Žikelić:

- K. Chatterjee, E. K. Goharshady, P. Novotný, and D. Žikelić. "Proving Non-Termination by Program Reversal". In: *Proceedings of PLDI'21*. Ed. by S. N. Freund and E. Yahav. ACM, 2021, pp. 1033–1048

First, we clarify the motivation for proving non-termination. While termination is the property typically desired to prove, if a termination proof fails, we do not know whether this is because the program does not terminate or whether our termination-proving technique was too weak. In such a case, it is useful to employ non-termination prover as a sort of *bug-hunting* procedure for termination: if the proof is successful, we can be sure that the program indeed contains an error, and the proof might help us localise and repair it. Hence, proving non-termination of programs is a highly active area of research [GHMRX08; CCFNO14; LNORR14; Gie+17; LH18; FG19; VR08].

Fix a program (represented as a non-probabilistic PTS with a state set S) with a given set of initial states. As discussed previously, a set $I \subseteq S$ is an *invariant* for the program if each state reachable from some initial state if contained in I. A set I is *inductive* if for every $s \in I$ the set I contains all one-step successors s' of s. Every inductive set that contains all initial states is an invariant, but not every invariant is inductive (since it might contain some unreachable state s but missing some successor of s). There exist many techniques for the synthesis of non-trivial (inductive) invariants. In our paper, we employ the technique of [GSV08] based on quadratic constraint solving.

To reason about non-termination, we introduce the notion of a *backward invariant (BI).*

Definition 10. *Consider a program with a set of terminal states F. A set of states BI is a* backward invariant *w.r.t. if every state s from which F is reachable is contained in BI. Moreover, we say that BI is* pre-inductive *if for every $s' \in BI$ and every one-step* predecessor *s of s' it holds $s \in BI$.*

A good way to conceptualise backward invariants is to imagine a program with all transitions "reversed." (I.e., if the original program contained a transition from state s to s', the reversed program contains a transition from s' to s. The reversed program has F as its set of initial states. Note that we work with non-probabilistic transitions). A backward invariant in the original program is then simply an invariant in the reversed program; and similarly, I is pre-inductive iff it is inductive in the reversed program.

The imperative arithmetic programs we consider can be translated into arithmetic transition systems, where each transition is represented as an arithmetical relation between the primed and unprimed variables. For these systems, the reversal can be done simply by swapping the primed and unprimed variables in the individual relations. We can then plug the reversed program into the aforementioned invariant synthesis technique to obtain pre-inductive backward invariants for the original program.

Backward invariants serve as sound certificates of non-termination:

Theorem 9. *Let $\mathcal{P}$ be a non-probabilistic program. If $\mathcal{P}$ admits a pre-inductive backward invariant BI such that BI is not an invariant of $\mathcal{P}$, then $\mathcal{P}$ admits a non-terminating execution.*

The theorem is easy to prove: since BI is not an invariant, its complement $\neg BI = S \setminus BI$ is reachable, so there exists an execution that enters $\neg BI$. Since BI is pre-inductive, $\neg BI$ is inductive, so the execution never leaves $\neg BI$ once it reaches it. Finally, since BI is a backward invariant, $\neg BI$ does not contain any terminal state, so the aforementioned execution is non-terminating.

We now sketch how to algorithmically find BI satisfying the conditions of Theorem 9. The problem is that using the constraint-based technique of [GSV08] cannot express the condition that BI is not an invariant. A naive solution to this issue would be to try to find any pre-inductive backward invariant and then check, using a safety prover, that its complement is reachable. However, if the check is only done after the synthesis, then nothing prevents the constraint solver from yielding the trivial backward invariant S, whose complement is empty. Hence, we separate the synthesis of BI into two separate steps. First, we check whether there exists a pre-inductive backward invariant BI which does not contain some initial state. This can be done using the constraint-based approach. Such BI, if it exists, is clearly not an invariant, so in the case of a positive answer, we can report non-termination without the need for a reachability check. Otherwise, we check if there exists a pre-inductive backward invariant BI which is not inductive. For such BI, we know that any execution that reaches $\neg BI$ is non-terminating, so we check whether the complement of BI is reachable using a safety checker. The idea is that by restricting the search for BI to non-inductive sets, the constraint-based approach will be less likely to return trivial solutions.

Our method is arguably simple, which we deem to be an advantage. Many technical details were omitted from the above description. In particular, our method also works for programs with non-determinism, which is not universally true for alternative approaches to non-termination proving. We use resolution of non-determinism through polynomial-template schedulers (synthesis of which can be incorporated into the constraint-solving step), which narrows down the set of possible behaviours and thus simplifies the search for a pre-inductive backward invariant. To summarise, our approach has several benefits which, to our best knowledge, are not *all* present in any alternative approach:

- it handles non-determinism;

- it handles polynomial-arithmetic (as opposed to only linear-arithmetic) programs;
- it is able to prove non-termination of programs in which all non-terminating executions are aperiodic (i.e. non-lasso);
- it comes with *relative completeness* guarantees (i.e. if the non-termination witness of a certain shape exists, our method is guaranteed to find it).

We evaluated our approach on benchmarks from the TermComp'19 termination-proving competition [GRSWY19]. With a proper configuration, our tool proved non-termination of more programs than any state-of-the art comparison tool, and even proved non-termination of 2 programs that were not proven by any other tool. The vast majority of non-terminations were proved using the first stage of the algorithm (*BI* not containing an initial configuration), though the second stage (with the safety prover) was also utilised for several programs, demonstrating its usefulness. The results demonstrate that very simple techniques can yield surprisingly good and efficient outcomes. It is worth noting, though, that these results were made possible by recent advances in SMT solving, which enabled us to solve the quadratic constraint systems efficiently.

DESIGN SAFETY: RISK-AWARE DECISION MAKING UNDER UNCERTAINTY

6

In this chapter, we focus on the design safety of decision-making algorithms for autonomous agents operating under statistical uncertainty. The prime model for such stochastic decision making are *Markov decision processes (MDPs),* which are essentially probabilistic transition systems (PTSs) as introduced in Chapter 2. In an MDP, the whole system (consisting of an agent and its environment) can be, in each time step, in one of many states, and in each step, the agent has the opportunity to choose from some set of *actions* enabled in the current state. Depending on the current state-action pair, the system probabilistically transitions into a new state, after which a new step begins, the process continuing in this fashion *ad infinitum.* By identifying actions with PTS transitions, we see that MDPs are indeed PTSs, with agent choices resolved by schedulers (which are called *strategies* or *policies* in the MDP setting). Hence, from now on we will use the terms PTSs/MDPs, transitions/actions, schedulers/policies, etc., interchangeably. In what follows, we will focus on MDPs with *finite* (though possibly large) state and actions spaces.

Similarly to the verification of PTSs, studied in previous chapters, MDPs are typically equipped with *objectives* the agent aims to satisfy. The major difference between the PTS and MDP "narratives" is that in PTS verification, we aim to show that the system satisfies a given property (objective) for every scheduler (adversarial/demonic view of non-determinism), while in MDPs we want to compute a single policy which makes the agent satisfy the objective (controllable/angelic view of nondeterminism). In particular, if the underlying objective is *quantitative* (i.e., each policy is assigned some number), the MDP approach is aimed at *optimisation* of this quantity rather than just verifying that the quantity surpasses or stays below some value for some or for every policy.

In a typical MDP, the task is to optimise the *expected payoff,* where the payoff is a suitable aggregation of per-step rewards. That is, each state-transition pair (s, τ) is assigned a numerical reward $r(s, \tau) \in \mathbb{R}$, and the step rewards $r_i = r(s_i, \tau_i)$ are aggregated over the whole run $\varrho = s_0\tau_0 s_1\tau_1 \dots s_{i-1}\tau_{i-1}s_i \dots$ using a suitable aggregation function. The aggregation can run either up to some finite *decision horizon* H or over the whole infinite run, in which case $H = \infty$. Aggregation functions most commonly encountered in the literature are the *discounted payoff,* defined as $disc_\gamma(\varrho) = \sum_{i=0}^{H} \gamma^i r_i$ for some discount factor $\gamma \in (0, 1)$; the *mean payoff* (also called *limit average*) $mp(\varrho) = \liminf_{i \to H} \frac{\sum_{j=0}^{i-1} r_j}{i}$; and the *total reward* $tot(\varrho) = \sum_{i=1}^{H} r_i$, which is well-defined whenever H is finite or the per-step rewards are non-negative. For a given aggregation function $payoff \in \{disc_\gamma, mp, tot\}$ the

task is then to find a policy σ maximising the expected payoff $\mathbb{E}_s^\sigma[\mathit{payoff}]$ from the initial state s.

The expected payoff maximisation in MDPs has been heavily studied from both mathematical [Put05] and computer-science [PT87] points of view. Over time, it became clear that the expected payoff is a rather crude optimisation criterion which is not suitable for every application scenario. For instance, a policy which always yields zero payoff is, under the expected payoff criterion, equivalent to the one which has two equiprobable outcomes: +100 and -100. The difference in the underlying payoff distribution is particularly important with respect to the agent's *risk-willingness:* a loss-averse agent would strictly prefer the constant policy over the second one. This reasoning introduced a notion of *risk* into MDP optimisation, and inevitably, various approaches to handling this notion eventually emerged. One approach replaces the classical expected payoff with various *risk metrics* of the payoff distribution, such as mean-variance, value at risk (*VaR*), or conditional value at risk (*CVaR,* also known as *expected shortfall*); see, e.g. [SLM12; PS12; Mai13; CGJP17; TGY20]. These approaches lead to single-dimensional optimisation, i.e. the goal is to minimise the risk metric. In this thesis, we focus on an alternative approach of *constrained optimisation,* where the agent aims to optimise the expected payoff subject to satisfying additional *risk constraints* on the policy.

The standard model of constrained probabilistic decision-making are *constrained MDPs (CMDPs)* [Alt99]. Here, apart from per-step rewards, each state-transition pair is assigned a per-step *penalty* $c(s,\tau)$. The penalties are again aggregated using a suitable function, and the task is to find a policy maximising the expected payoff subject to the constraint that the expected aggregated penalty is below a given threshold. (The standard CMPD model allows multiple penalty functions, each with a separate threshold. For simplicity, we consider only single-dimensional penalties.) Various approaches to solving CMDPs were developed [Alt99; UH10; Pou+15; STW16], some of them for the more general partially observable setting (which we will discuss in more detail later on). While these works laid the theoretical groundwork for tackling the problem, the algorithms presented therein (based on classical planning techniques such as branch-and-bound or linear programming) do not match the performance of modern heuristic and learning algorithms for unconstrained payoff optimisation. Only relatively recently have the techniques such as *policy gradient* and *Monte Carlo tree search* been started to be applied to CMDPs with both perfect and partial observation [CGJP17; LKPK18]. The work of the thesis's author was a part of that trend. In the remainder of this chapter, we will present three publications in which the author has aimed to achieve the following objectives:

- solve constraint optimisation problems for MDPs in which the constraint does not directly fit into the CMDP framework;
- design algorithms that are suitable for solving models with both perfect and partial observation; and
- design algorithms that scale well to very large models.

6.1 POMDP Optimisation under Worst-Case Payoff Constraints

We start with the following AAAI'17 paper co-authored with Chatterjee, Pérez, Raskin, and Žikelić:

- K. Chatterjee, P. Novotný, G. A. Pérez, J. Raskin, and D. Žikelić. "Optimizing Expectation with Guarantees in POMDPs". In: *Proceedings of AAAI'17*. AAAI Press, 2017, pp. 3725–3732. URL: http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14354

The paper considers optimisation in *partially observable* MDPs (*POMDPs*). In a POMDP, the agent cannot directly observe the current state. Instead, once entering some state s, the state emits an *observation* o according to a fixed state-dependent distribution O_s. The agent receives the emitted observation and incorporates it in its decision. For the sake of the latter, we assume that there is some global set of transition labels called *actions*, with transitions enabled in different states possibly sharing an action label. Making a decision then means selecting an action a to "play," which results in performing a transition that is enabled in the current state and labelled with a. In line with this reasoning, the policies in POMDPs output distributions over actions and do not input state-transition histories $s_0\tau_0 s_1\tau_1 \dots \tau_{i-1}s_i$, but observation-action histories $o_0a_0o_1a_1 \dots a_{i-1}o_i$ where $o_j \sim O_{s_j}$ and a_j is the label of transition τ_j. In what follows, we also assume that the agent can observe the step rewards, i.e. that these rewards can be framed as a function of the current observation and the selected action.

In the presented paper, we consider the problem of optimising the expected discounted payoff under the constraint that no run emitted by the policy has discounted payoff smaller than a given threshold b. That is, we are solving the problem

$$\max_{\sigma} \mathbb{E}^{\sigma}_{s_0}[disc_\gamma]$$
$$\text{subject to } \mathbb{P}^{\sigma}_{s_0}[disc_\gamma < b] = 0.$$

(For discounted payoff, it can be proved that almost-sure satisfaction of the constraint is equivalent to its sure satisfaction.) This corresponds to the problem of *beyond worst case (BWC)* optimisation in perfectly observable MDPs [BFRR14; RRS15]. However, utilising the previous BWC approaches would first require us to exactly solve the unconstrained optimisation problem. While this is, in principle, possible in MDPs, where polynomial-time algorithms for the unconstrained problem are known, solving POMDPs exactly is generally considered to be intractable. Instead, various heuristic and statistical approaches are the method of choice for POMDP solving.

To take this into account, we take the following approach to the problem: take some efficient heuristic algorithm for unconstrained POMDPs and modify it so as to never select

actions that might lead to violation of the constraints. The idea is similar to the concepts of *permissive controller synthesis* [DFKPU14] and *shielding* [Als+18] in MDPs, which were emerging at the same time as our publication. In the terminology of the latter, our approach can be explained as consisting of two phases:

- In the first — offline — phase, we compute a *shield:* an algorithm which inputs the POMDP history (or some characteristic thereof) and outputs a list of *allowed actions*. The shield has the property that as long as the agent plays only actions allowed in respective steps, the constraint is never violated. Conversely, no safe (i.e. constraint-satisfying) policy is prohibited by the shield: once the agent plays a disallowed action, he is guaranteed to eventually violate the constraint.
- Then, we run some state-of-the-art algorithm for unconstrained POMDP optimisation augmented with the shield. The shield keeps track of the agent's history and prohibits the usage of disallowed actions by the algorithm.

We briefly elaborate on our approach to the two aforementioned points. We prove that the shield only needs to keep track of the payoff accumulated so far and of the agent's *belief support:* the set of states in which the POMDP can currently be with non-zero probability. Belief supports are uniquely determined by the current history and can be efficiently updated from step to step. To compute the actions allowed for each payoff-belief support pair, we solve a finite turn-based zero-sum discounted-payoff game whose states correspond to possible belief supports. One player in the game corresponds to the agent, whereas the other resolves the probabilistic behaviour of the environment in an adversarial way. (The constraint in our problem is such that exact transition probabilities do not matter. What matters are the worst possible outcomes of the probabilistic choices.) This way of computing the shield might seem inefficient: there is no known polynomial-time algorithm for discounted-payoff games (though they are known to be in NP ∩ coNP [ZP96]) and moreover, the number of belief supports can be exponentially large in the size of the game. However, in practice, the number of belief supports reachable from the initial setup could be much lower due to regularities in the POMDP's structure. As for solving the resulting game, the classical *value iteration* algorithm [Put05] proved to work very well in our experiments.

For the second, optimisation phase, we utilised the POMCP algorithm [SV10], which extends the *Monte Carlo tree search* (*MCTS*) planning algorithm for MDPs [KS06]. MCTS can be described as a heuristic search algorithm exploring a part of the infinite tree of all of the model's histories (a *history tree*). It is an online algorithm which, in every decision step, computes a local approximation of the optimal policy so as to select the best action for the current situation. To achieve this, it iteratively builds a *search tree* – a finite connected sub-graph of the history tree, whose each node carries statistical information about the best payoff achievable by histories passing through that node. In each iteration, the current history (representing the root of the search tree) is extended with a finite suffix sampled according to a two-step process:

- As long as the suffix is in the current search tree, it is being extended by selecting actions according to the UCT formula [KS06], which balances the exploration of new histories with the exploitation of promising ones. After each action selection, the next observation is then sampled according to the selected action and dynamics of the given POMDP.
- Once the suffix falls out of the search tree, the process continues similarly but with actions selected uniformly at random (so-called *rollout phase*).

After the suffix is sampled, its shortest prefix not yet included in the search tree is added to the tree, and all nodes on the path from the newly added one to the tree root have their statistical information updated by the payoff of the sampled history. After a number of such iterations, the statistical information in the root is used to estimate the optimal action in the current step. This action is then performed, and a new observation is received, corresponding to one successor node v of the root node. This v then becomes a new root of the tree (i.e., all nodes outside of v's sub-tree are pruned away), and the resulting tree is utilised in the next decision epoch.

We extended POMCP by storing, in each node, the information about the past accumulated payoff and the current belief support. During the building of the search tree and final action selection, this information is passed to the shield so as to retrieve the list of actions from which POMCP can sample. We demonstrated the effectiveness of our approach on several classical POMDP benchmarks with up to $\approx$ 10,000 states.

6.2 From Worst-Case to Quantile Constraints

We continue with the following IJCAI-ECAI'18 paper co-authored with Chatterjee, Elgyütt, and Rouillé:

- K. Chatterjee, A. Elgyütt, P. Novotný, and O. Rouillé. "Expectation Optimization with Probabilistic Guarantees in POMDPs with Discounted-Sum Objectives". In: *Proceedings of IJCAI'18*. Ed. by J. Lang. ijcai.org, 2018, pp. 4692–4699. DOI: 10.24963/ijcai.2018/652. URL: https://doi.org/10.24963/ijcai.2018/652

In the paper, we generalised the objective from the previous section to allow constraint violations with some given (ideally, small) probability. That is, apart from the payoff threshold b, we are also given a *risk threshold* θ and solve the following problem:

$$\max_{\sigma} \mathbb{E}^{\sigma}_{s_0}[disc_{\gamma}]$$
$$\text{subject to } \mathbb{P}^{\sigma}_{s_0}[disc_{\gamma} < b] \leq \theta.$$

The rationale behind this objective is that in some application domains, satisfying the worst-case constraint for non-trivial values of b might be impossible, since the agent could always experience some catastrophic failure. Hence, we relax the constraints by allowing a bounded probability of large shortfalls. This "quantile" constraint in the problem's formulation is conceptually similar to the notion of *value-at-risk* utilised in mathematical finance [DP97].

It is not possible to straightforwardly adapt the shielding approach from the previous section to this more general scenario due to several factors. First, even deciding whether there exists a policy satisfying a given quantile constraint in a *perfectly observable MDP* has been proved to be highly intractable for several classes of payoff functions [BCFNS13; HK15]. Second, optimal policies for the quantile-constrained problem can be randomised: the agent might want to probabilistically mix risky but potentially high-yield actions with conservative low-yield actions so as to keep the actual risk (i.e. the probability that $disc_\gamma < b$) as close to θ as possible (otherwise he would be sacrificing the optimality of expected payoff). Hence, the shield would need to disable *mixtures* of actions rather than individual actions. Third, the agent's risk threshold θ actually evolves during the decision-making process. To illustrate this, imagine that the agent, whose risk threshold is θ, has chosen a risky action with two equiprobable outcomes, one of which eventually leads to payoff that is surely below b. If such a bad outcome indeed happens, there is nothing to be done to save the agent, but if the agent is lucky and observes the alternative outcome, he needs to adjust his risk threshold: before the action outcome has been observed, the probability of violating the payoff constraint was $\frac{1}{2} + \frac{1}{2}p$, where p is the eventual constraint violation probability in the alternative outcome. To ensure that this probability is at most θ, we must have $p \leq 2\theta - 1$, i.e. the right-hand side is the new risk threshold for the continuation of the process. Hence, the shields would additionally need to input the current level of θ, which further complicates their computation.

Our solution to the aforementioned conundrum is to eschew formal guarantees and instead propose a heuristic approach to constrained optimisation, which is only guaranteed to find a constraint-satisfying policy in the limit. We dispose of the shield and instead directly modify POMCP to obtain a new algorithm called RAMCP ("Risk-aware POMCP"). Our modification can be described as follows:

- Whenever a history is sampled, its payoff is compared against the payoff threshold b. If the history has payoff at least p, we assign it *risk value* 0, otherwise its risk value is 1. The statistics of the risk values are stored in the nodes of the search tree analogously to the payoff statistics.

- After the search three extension phase stops and the algorithm is about to select the actual action to play, we proceed as follows: we interpret the search tree as a *perfectly observable constrained MDP* with nodes as states and edges as transitions. To compute the transition probabilities, we compute *beliefs* corresponding to each particular node, i.e. probability distribution over the states of the POMDP indicating, for each state,

the probability of being in that particular state given that the history corresponding to the tree node was observed. The belief computation proceeds by performing standard Bayesian updates [SV10] along the tree edges. Inside the tree, the per-step reward function over edges is computed in a similar way from the reward function of the underlying POMDP, while the penalty of every tree edge is zero. Finally, for each leaf node of the tree, we incur one-time reward and penalty equal to the respective statistical estimates stored within the node (formally, this is done by adding dummy transitions to a special sink state). The resulting CMDP with the tree root as the initial state is solved via the usual linear programming approach [Alt99], which in particular yields a probability distribution over actions in the root. The action to be played is sampled according to this distribution, and a resulting observation is received. Thereafter, the tree is pruned similarly to POMCP and the agent's risk threshold is updated in the manner sketched above. The algorithm then proceeds to the next decision epoch.

In essence, we use the POMCP sampling to sample a finite CMDP which forms a local approximation of the overall partially observable process, with "distant" behaviours estimated statistically in the leaf nodes. It is clear that this heuristic cannot be equipped with concrete formal guarantees. (And moreover, since the POMCP sampling is inherently finite-horizon, RAMCP solves only a finite-horizon approximation of the problem.) However, one can prove that as the number of search tree iterations approaches infinity, the algorithm indeed converges to the optimal constrained action. Moreover, we validated the algorithm on a set of standard POMDP benchmarks with up to 67000 states. In our experiments, the agent's risk never surpassed the initial risk threshold θ, and hence the algorithm behaved in a conservative way.

6.3 Risk-Constrained Learning

We conclude the overview with the following AAAI'20 paper co-authored with Brázdil, Chatterjee, and Vahala:

- T. Brázdil, K. Chatterjee, P. Novotný, and J. Vahala. "Reinforcement Learning of Risk-Constrained Policies in Markov Decision Processes". In: *The Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI 2020)*. AAAI Press, 2020, pp. 9794–9801. URL: https://aaai.org/ojs/index.php/AAAI/article/view/6531

The aim of the paper was to scale the algorithm presented in the previous section to even larger state spaces by incorporating elements of machine learning. Since we anticipated non-trivial engineering challenges (in particular w.r.t. setting of hyperparameters), we simplified the problem in two ways:

- We restricted to the perfectly observable setting, i.e. considered MDPs only; and

- we considered simpler *state-based risk* constraints. That is, the agent was given a set F of *failure states* to avoid and a risk threshold θ; his task was then to find a policy maximising the expected payoff while ensuring that the probability of reaching F is below *theta*:

$$\max_{\sigma} \mathbb{E}^{\sigma}_{s_0}[disc_{\gamma}]$$
$$\text{subject to } \mathbb{P}^{\sigma}_{s_0}[F \text{ is reached}] \leq \theta.$$

Since we stick to the MCTS approach to solving the problem, simplifying the constraint does not significantly decrease its expressive power: as seen in the previous section, e.g. quantile constraints can be expressed using state-based risk by extending each history node with the information on the payoff accumulated so far. Also, while perfectly observable CMDPs are known to be solvable by linear programming [Alt99], and thus in polynomial time, our goal was to provide an algorithm that would be applicable to very large state spaces, in which already writing down the whole model, not least constructing the linear program over it, would be computationally demanding.

The core idea behind our approach is taking RAMCP and replacing the rollout-based estimation of payoffs and risks with an estimation via a *predictor* (*function approximator*) that is iteratively learning the correct values by observing repeated agent-environment simulations. The resulting algorithm is called RAlph, since the predictor is used in a way similar to the well-known MCTS-based AlphaZero game-playing algorithm [Sil+18]. RAlph combines the basic structure of AlphaZero with the RAMCP approach of sampling a local constrained MDP to be solved by linear programming.

While the conceptual idea behind RAlph is a simple-looking combination of known algorithms, the major difficulty in RAlph's development was the engineering aspect, i.e. designing the whole learning pipeline so that the learning process is stable and scalable. There were several degrees of freedom in the design of RAlph's specific components that needed to be investigated properly so as to make the correct choices. In the end, RAlph has been shown to handle MDPs with up to millions of states, and even on smaller benchmarks, it clearly outperformed RAMCP by a wide margin. The performance of RAlph can be further increased by automated parameter tuning [Pet22].

NOTES

1. It might be argued that there was no real randomness used, just pseudo-random generators. However, we do not aim to delve into debates on the nature of randomness in this thesis. If an observer cannot efficiently distinguish the output of some procedure from a random output, we deem the procedure to be randomised.
2. We understand the phrase *probabilistic program analysis* as a shorthand for *program analysis for probabilistic programs.* I.e., the phrase does not imply that the analysis algorithms we develop are themselves probabilistic.
3. For continuous PTSs, the scheduler needs to satisfy an additional *measurability* condition for the semantics to be well-defined [NSK09].
4. In this work, we always consider *mathematical* numbers as opposed to machine ones.
5. Since the capabilities of modern SMT solvers have increased significantly over the last decade, SMT solving should be considered as an efficient alternative for tackling quadratic systems.
6. We denote by $\mathbb{E}_{x\sim\mathcal{D}}[\ldots]$ the expected value in a random experiment consisting of sampling x according to the distribution $\mathcal{D}$.
7. For ergodic systems, the i.i.d. behaviour can sometimes be recovered by considering a "big-step" process in which one time step corresponds to returning to some distinguished state. But the ergodic assumption might be too strong in itself. Nondeterminism further complicates the attempts to proceed in this direction.
8. The concept of an invariant supporting an RSM is defined analogously to an invariant supporting a ranking function.

BIBLIOGRAPHY

[ACN18] S. Agrawal, K. Chatterjee, and P. Novotný. "Lexicographic ranking supermartingales: an efficient approach to termination of probabilistic programs". In: *Proc. ACM Program. Lang.* 2. (Proceedings of POPL'18) (2018), 34:1–34:32. DOI: 10.1145/3158122. URL: https://doi.org/10.1145/3158122.

[ADFG10] C. Alias, A. Darte, P. Feautrier, and L. Gonnord. "Multi-dimensional Rankings, Program Termination, and Complexity Bounds of Flowchart Programs". In: *Proceedings of the 17th International Conference on Static Analysis*. Ed. by R. Cousot and M. Martel. Perpignan, France: Springer-Verlag, 2010, pp. 117–133. URL: http://dl.acm.org/citation.cfm?id=1882094.1882102.

[All08] L. J. S. Allen. "An Introduction to Stochastic Epidemic Models". In: *Mathematical Epidemiology*. Ed. by F. Brauer, P. van den Driessche, and J. Wu. Springer, 2008, pp. 81–130.

[Als+18] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu. "Safe Reinforcement Learning via Shielding". In: *AAAI Conference on Artificial Intelligence, (AAAI 2018)*. Ed. by S. A. McIlraith and K. Q. Weinberger. AAAI Press, 2018, pp. 2669–2678. URL: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17211.

[Alt99] E. Altman. *Constrained Markov Decision Processes*. Chapman&Hall/CRC, 1999.

[BBE10] T. Brázdil, V. Brožek, and K. Etessami. "One-Counter Stochastic Games". In: *Proceedings of FSTTCS'10*. Ed. by K. Lodaya and M. Mahajan. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2010, pp. 108–119.

[BBEK11] T. Brázdil, V. Brožek, K. Etessami, and A. Kučera. "Approximating the termination value of one-counter MDPs and stochastic games". In: *Proceedings of ICALP 2011, Part II*. Vol. 6756. 2011, pp. 332–343.

[BBEKW10] T. Brázdil, V. Brožek, K. Etessami, A. Kučera, and D. Wojtczak. "One-Counter Markov Decision Processes". In: *Proceedings of SODA 2010*. 2010, pp. 863–874.

[BBKO10] T. Brázdil, V. Brožek, A. Kučera, and J. Obdržálek. "Qualitative Reachability in Stochastic BPA Games". In: *Information and Computation* 208.7 (2010), pp. 772–796.

[BCFNS13] T. Brázdil, T. Chen, V. Forejt, P. Novotný, and A. Simaitis. "Solvency Markov Decision Processes with Interest". In: *Proceedings of IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2013)*. Ed. by A. Seth and N. K. Vishnoi. Vol. 24. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013, pp. 487–499. DOI: 10.4230/LIPIcs.FSTTCS.2013.487. URL: http://dx.doi.org/10.4230/LIPIcs.FSTTCS.2013.487.

[BCIKP16] M. Brockschmidt, B. Cook, S. Ishtiaq, H. Khlaaf, and N. Piterman. "T2: Temporal Property Verification". In: *Tools and Algorithms for the Construction and Analysis of Systems: 22nd International Conference, TACAS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*. Ed. by M. Chechik and J.-F. Raskin. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 387–393. DOI: 10.1007/978-3-662-49674-9_22. URL: http://dx.doi.org/10.1007/978-3-662-49674-9%5C_22.

[BCNV20] T. Brázdil, K. Chatterjee, P. Novotný, and J. Vahala. "Reinforcement Learning of Risk-Constrained Policies in Markov Decision Processes". In: *The Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI 2020)*. AAAI Press, 2020, pp. 9794–9801. URL: https://aaai.org/ojs/index.php/AAAI/article/view/6531.

[BEKK12] T. Brázdil, J. Esparza, S. Kiefer, and A. Kučera. "Analyzing Probabilistic Pushdown Automata". In: *Formal Methods in System Design* 43.2 (2012), pp. 124–163.

[BFRR14] V. Bruyère, E. Filiot, M. Randour, and J.-F. Raskin. "Meet Your Expectations With Guarantees: Beyond Worst-Case Synthesis in Quantitative Games". In: *Proceedings of STACS'14*. Ed. by E. W. Mayr and N. Portier. Vol. 25. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2014, pp. 199–213. DOI: 10.4230/LIPIcs.STACS.2014.199. URL: http://dx.doi.org/10.4230/LIPIcs.STACS.2014.199.

[BG05] O. Bournez and F. Garnier. "Proving Positive Almost-Sure Termination". In: *Proceedings of the 16th International Conference on Rewriting Techniques and Applications (RTA 2005)*. Ed. by J. Giesl. 3467 vols. LNCS. Springer, 2005, pp. 323–337.

[BG15] A. M. Ben-Amram and S. Genaim. "Complexity of Bradley-Manna-Sipma Lexicographic Ranking Functions". In: *Computer Aided Verification: 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part II*. Ed. by D. Kroening and C. S. Păsăreanu. Springer International Publishing, 2015, pp. 304–321. DOI: 10.1007/978-3-319-

21668-3_18. URL: http://dx.doi.org/10.1007/978-3-319-21668-3_18.

[Bil95] P. Billingsley. *Probability and Measure*. 3rd. Wiley, 1995.

[BK08] C. Baier and J.-P. Katoen. *Principles of Model Checking*. Cambridge, Massachusetts, 2008, p. 984.

[BKK14] T. Brázdil, S. Kiefer, and A. Kučera. "Efficient Analysis of Probabilistic Programs with an Unbounded Counter". In: *J. ACM* 61.6 (Dec. 2014), 41:1–41:35. DOI: 10.1145/2629599. URL: http://doi.acm.org/10.1145/2629599.

[BKKN15] T. Brázdil, S. Kiefer, A. Kučera, and P. Novotný. "Long-Run Average Behaviour of Probabilistic Vector Addition Systems". In: *Proceedings of LICS'15*. IEEE Computer Society, 2015, pp. 44–55.

[BKKNK14] T. Brázdil, S. Kiefer, A. Kučera, P. Novotný, and J.-P. Katoen. "Zero-Reachability in Probabilistic Multi-Counter Automata". In: *Proceedings of CSL-LICS'14*. Vienna, Austria: ACM, 2014, 22:1–22:10. DOI: 10.1145/2603088.2603161. URL: http://doi.acm.org/10.1145/2603088.2603161.

[BKNW12] T. Brázdil, A. Kučera, P. Novotný, and D. Wojtczak. "Minimizing Expected Termination Time in One-Counter Markov Decision Processes". In: *Proceedings of ICALP 2012, Part II*. Vol. 7392. 2012, pp. 141–152.

[BMS05] A. R. Bradley, Z. Manna, and H. B. Sipma. "Linear Ranking with Reachability". In: *Computer Aided Verification, 17th International Conference, CAV 2005, Edinburgh, Scotland, UK, July 6-10, 2005, Proceedings*. Ed. by K. Etessami and S. K. Rajamani. Springer, 2005, pp. 491–504. DOI: 10.1007/11513988_48.

[CCFNO14] H.-Y. Chen, B. Cook, C. Fuhs, K. Nimkar, and P. O'Hearn. "Proving Nontermination via Safety". In: *Tools and Algorithms for the Construction and Analysis of Systems: 20th International Conference, TACAS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014. Proceedings*. Ed. by E. Ábrahám and K. Havelund. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 156–171. DOI: 10.1007/978-3-642-54862-8_11. URL: http://dx.doi.org/10.1007/978-3-642-54862-8_11.

[CENR18] K. Chatterjee, A. Elgyütt, P. Novotný, and O. Rouillé. "Expectation Optimization with Probabilistic Guarantees in POMDPs with Discounted-Sum Objectives". In: *Proceedings of IJCAI'18*. Ed. by J. Lang. ijcai.org, 2018, pp. 4692–4699. DOI: 10.24963/ijcai.2018/652. URL: https://doi.org/10.24963/ijcai.2018/652.

[CF17] K. Chatterjee and H. Fu. *Termination of Nondeterministic Recursive Probabilistic Programs*. 2017. DOI: 10.48550/ARXIV.1701.02944. URL: https://arxiv.org/abs/1701.02944.

[CFG16] K. Chatterjee, H. Fu, and A. K. Goharshady. "Termination Analysis of Probabilistic Programs Through Positivstellensatz's". In: *Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part I*. Ed. by S. Chaudhuri and A. Farzan. Vol. 9779. Lecture Notes in Computer Science. Springer, 2016, pp. 3–22. DOI: 10.1007/978-3-319-41528-4_1. URL: https://doi.org/10.1007/978-3-319-41528-4%5C_1.

[CFN20] K. Chatterjee, H. Fu, and P. Novotný. "Termination Analysis of Probabilistic Programs with martingales". In: *Foundations of Probabilistic Programming*. Ed. by G. Barthe, J.-P. Katoen, and A. Silva. Cambridge University Press, 2020. Chap. 7, pp. 221–258.

[CFNH16] K. Chatterjee, H. Fu, P. Novotný, and R. Hasheminezhad. "Algorithmic analysis of qualitative and quantitative termination problems for affine probabilistic programs". In: *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*. Ed. by R. Bodík and R. Majumdar. ACM, 2016, pp. 327–342. DOI: 10.1145/2837614.2837639. URL: http://doi.acm.org/10.1145/2837614.2837639.

[CFNH18] K. Chatterjee, H. Fu, P. Novotný, and R. Hasheminezhad. "Algorithmic Analysis of Qualitative and Quantitative Termination Problems for Affine Probabilistic Programs". In: *ACM Trans. Program. Lang. Syst.* 40.2 (2018), 7:1–7:45. DOI: 10.1145/3174800. URL: https://doi.org/10.1145/3174800.

[CGJP17] Y. Chow, M. Ghavamzadeh, L. Janson, and M. Pavone. "Risk-Constrained Reinforcement Learning with Percentile Risk Criteria". In: *Journal of Machine Learning Research* 18.1 (2017), pp. 6070–6120.

[CGNŽ21] K. Chatterjee, E. K. Goharshady, P. Novotný, and D. Žikelić. "Proving Non-Termination by Program Reversal". In: *Proceedings of PLDI'21*. Ed. by S. N. Freund and E. Yahav. ACM, 2021, pp. 1033–1048.

[CGNZŽ21] K. Chatterjee, E. K. Goharshady, P. Novotný, J. Zárevúcky, and D. Žikelić. "On Lexicographic Proof Rules for Probabilistic Termination". In: *Proceedings of FM'21*. Ed. by M. Huisman, C. S. Pasareanu, and N. Zhan. Vol. 13047. LNCS. Springer, 2021, pp. 619–639.

[CNPRŽ17] K. Chatterjee, P. Novotný, G. A. Pérez, J. Raskin, and D. Žikelić. "Optimizing Expectation with Guarantees in POMDPs". In: *Proceedings of AAAI'17*. AAAI Press, 2017, pp. 3725–3732. URL: http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14354.

[CNŽ17] K. Chatterjee, P. Novotný, and D. Žikelić. "Stochastic invariants for probabilistic termination". In: *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL 2017)*. Ed. by G. Castagna and A. D. Gordon. ACM, 2017, pp. 145–160. DOI: 10.1145/3009837. URL: http://dl.acm.org/citation.cfm?id=3009873.

[CPR06] B. Cook, A. Podelski, and A. Rybalchenko. "Termination Proofs for Systems Code". In: *Proceedings of the ACM SIGPLAN 2006 Conference on Programming Language Design and Implementation, Ottawa, Ontario, Canada, June 11-14, 2006*. Ed. by M. I. Schwartzbach and T. Ball. ACM, 2006, pp. 415–426. DOI: 10.1145/1133981.1134029. URL: https://doi.org/10.1145/1133981.1134029.

[CPR11] B. Cook, A. Podelski, and A. Rybalchenko. "Proving program termination". In: *Communications of the ACM* 54.5 (2011), pp. 88–98. DOI: 10.1145/1941487.1941509. URL: http://doi.acm.org/10.1145/1941487.1941509.

[CS01] M. Colón and H. Sipma. "Synthesis of Linear Ranking Functions". In: *Tools and Algorithms for the Construction and Analysis of Systems, 7th International Conference, TACAS 2001 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2001 Genova, Italy, April 2-6, 2001, Proceedings*. Ed. by T. Margaria and W. Yi. Springer, 2001, pp. 67–81. DOI: 10.1007/3-540-45319-9_6.

[CS02] M. A. Colón and H. B. Sipma. "Practical Methods for Proving Program Termination". In: *Computer Aided Verification: 14th International Conference, CAV 2002 Copenhagen, Denmark, July 27–31, 2002 Proceedings*. Ed. by E. Brinksma and K. G. Larsen. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 442–454. DOI: 10.1007/3-540-45657-0_36. URL: http://dx.doi.org/10.1007/3-540-45657-0_36.

[CS13] A. Chakarov and S. Sankaranarayanan. "Probabilistic Program Analysis with Martingales". In: *Computer Aided Verification - 25th International Conference (CAV 2013)*. Ed. by N. Sharygina and H. Veith. Vol. 8044. LNCS. Springer, 2013, pp. 511–526. DOI: 10.1007/978-3-642-39799-8_34.

[CSS03] M. A. Colón, S. Sankaranarayanan, and H. B. Sipma. "Linear invariant Generation Using Non-Linear Constraint Solving". In: *International Conference on Computer Aided Verification*. Springer. 2003, pp. 420–432.

[CSZ13] B. Cook, A. See, and F. Zuleger. "Ramsey vs. Lexicographic Termination Proving". In: *TACAS*. Ed. by N. Piterman and S. A. Smolka. Springer, 2013, pp. 47–61.

[DFKPU14] K. Dräger, V. Forejt, M. Kwiatkowska, D. Parker, and M. Ujma. "Permissive Controller Synthesis for Probabilistic Systems". In: *Proceedings of TACAS'14*. Ed. by E. Ábrahám and K. Havelund. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 531–546.

[Dij76] E. W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976.

[DM79] N. Dershowitz and Z. Manna. "Proving Termination with Multiset Orderings". In: *Proceedings of ICALP'79*. Ed. by H. A. Maurer. Berlin, Heidelberg: Springer, 1979, pp. 188–202.

[DP97] D. Duffie and J. Pan. "An overview of value at risk." In: *The Journal of Derivatives* 4.3 (1997), pp. 7–49.

[ESY12] K. Etessami, A. Stewart, and M. Yannakakis. "Polynomial Time Algorithms for Branching Markov Decision Processes and Probabilistic Min(Max) Polynomial Bellman Equations". English. In: *Automata, Languages, and Programming 2012*. Ed. by A. Czumaj, K. Mehlhorn, A. Pitts, and R. Wattenhofer. Vol. 7391. LNCS. Springer Berlin Heidelberg, 2012, pp. 314–326. DOI: 10.1007/978-3-642-31594-7_27. URL: http://dx.doi.org/10.1007/978-3-642-31594-7_27.

[ESY17] K. Etessami, A. Stewart, and M. Yannakakis. "A Polynomial Time Algorithm for Computing Extinction Probabilities of Multitype Branching Processes". In: *SIAM J. Comput.* 46.5 (2017), pp. 1515–1553. DOI: 10.1137/16M105678X. URL: https://doi.org/10.1137/16M105678X.

[EY12] K. Etessami and M. Yannakakis. "Model Checking of Recursive Probabilistic Systems". In: 13 (2 2012).

[Fel84] Y. A. Feldman. "A decidable propositional dynamic logic with explicit probabilities". In: *Information and Control* 63.1 (1984), pp. 11–38. DOI: 10.1016/S0019-9958(84)80039-X. URL: http://www.sciencedirect.com/science/article/pii/S001999588480039X.

[FG19] F. Frohn and J. Giesl. "Proving Non-Termination via Loop Acceleration". In: *2019 Formal Methods in Computer Aided Design, FMCAD 2019, San Jose, CA, USA, October 22-25, 2019*. 2019, pp. 221–230.

[FH15] L. M. F. Fioriti and H. Hermanns. "Probabilistic Termination: Soundness, Completeness, and Compositionality". In: *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2015)*. Ed. by S. K. Rajamani and D. Walker. ACM, 2015, pp. 489–501. DOI: 10.1145/2676726.2677001.

[FH82] Y. A. Feldman and D. Harel. "A probabilistic dynamic logic". In: *Proceedings of the fourteenth annual ACM Symposium on Theory of computing*. ACM. 1982, pp. 181–195.

[Flo67] R. W. Floyd. "Assigning meanings to programs". In: *Mathematical Aspects of Computer Science* 19 (1967), pp. 19–33.

[Fos53] F. G. Foster. "On the Stochastic Matrices Associated with Certain Queuing Processes". In: *The Annals of Mathematical Statistics* 24.3 (1953), pp. 355–360.

[GHMRX08] A. Gupta, T. A. Henzinger, R. Majumdar, A. Rybalchenko, and R.-G. Xu. "Proving Non-termination". In: *SIGPLAN Not.* 43.1 (Jan. 2008), pp. 147–158. DOI: 10.1145/1328897.1328459. URL: http://doi.acm.org/10.1145/1328897.1328459.

[Gie+17] J. Giesl, C. Aschermann, M. Brockschmidt, F. Emmes, F. Frohn, C. Fuhs, J. Hensel, C. Otto, M. Plücker, P. Schneider-Kamp, T. Ströder, S. Swiderski, and R. Thiemann. "Analyzing Program Termination and Complexity Automatically with AProVE". In: *J. Autom. Reason.* 58.1 (2017), pp. 3–31. DOI: 10.1007/s10817-016-9388-y. URL: https://doi.org/10.1007/s10817-016-9388-y.

[Gri03] D. Gries. "Loop Invariant". In: *Encyclopedia of Computer Science*. GBR: John Wiley and Sons Ltd., 2003, pp. 1038–1040.

[GRSWY19] J. Giesl, A. Rubio, C. Sternagel, J. Waldmann, and A. Yamada. "The Termination and Complexity Competition". In: *Tools and Algorithms for the Construction and Analysis of Systems - 25 Years of TACAS: TOOLympics, Held as Part of ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part III*. 2019, pp. 156–166. DOI: 10.1007/978-3-030-17502-3_10. URL: https://doi.org/10.1007/978-3-030-17502-3%5C_10.

[GSV08] S. Gulwani, S. Srivastava, and R. Venkatesan. "Program analysis as constraint solving". In: *Proceedings of the ACM SIGPLAN 2008 Conference on Programming Language Design and Implementation, Tucson, AZ, USA, June 7-13, 2008*. 2008, pp. 281–292.

[HFCG19] M. Huang, H. Fu, K. Chatterjee, and A. K. Goharshady. "Modular Verification for Almost-Sure Termination of Probabilistic Programs". In: *Proc. ACM Program. Lang.* 3.OOPSLA (Oct. 2019). DOI: 10.1145/3360555. URL: https://doi.org/10.1145/3360555.

[HK15] C. Haase and S. Kiefer. "The Odds of Staying on Budget". In: *International Colloquium on Automata, Languages, and Programming, (ICALP 2015)*. Ed. by M. M. Halldórsson, K. Iwama, N. Kobayashi, and B. Speckmann. Vol. 9135. Lecture Notes in Computer Science. Springer, 2015, pp. 234–246. DOI: 10.1007/978-3-662-47666-6_19. URL: https://doi.org/10.1007/978-3-662-47666-6%5C_19.

[HWZ08] H. Hermanns, B. Wachter, and L. Zhang. "Probabilistic CEGAR". In: *CAV*. LNCS 5123. Springer, 2008, pp. 162–175.

[KKNP09] M. Kattenbelt, M. Kwiatkowska, G. Norman, and D. Parker. "Abstraction refinement for probabilistic software". In: *International Workshop on Verification, Model Checking, and Abstract Interpretation*. Springer. 2009, pp. 182–197.

[KM75] S. Katz and Z. Manna. "A Closer Look at Termination". In: *Acta Informatica* 5 (1975), pp. 333–352. DOI: 10.1007/BF00264565. URL: https://doi.org/10.1007/BF00264565.

[KNP06] M. Z. Kwiatkowska, G. Norman, and D. Parker. "Game-based Abstraction for Markov Decision Processes". In: *QEST*. 2006, pp. 157–166.

[Kön27] D. König. "Über eine Schlussweise aus dem Endlichen ins Unendliche". In: *Acta Sci. Math.(Szeged)* 3.2-3 (1927), pp. 121–130.

[Koz83] D. Kozen. "A Probabilistic PDL". In: *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*. STOC '83. New York, NY, USA: ACM, 1983, pp. 291–297. DOI: 10.1145/800061.808758. URL: http://doi.acm.org/10.1145/800061.808758.

[KP12] M. Z. Kwiatkowska and D. Parker. "Advances in Probabilistic Model Checking". In: *Software Safety and Security - Tools for Analysis and Verification*. Ed. by T. Nipkow, O. Grumberg, and B. Hauptmann. Vol. 33. NATO Science for Peace and Security Series - D: Information and Communication Security. IOS Press, 2012, pp. 126–151. DOI: 10.3233/978-1-61499-028-4-126. URL: https://doi.org/10.3233/978-1-61499-028-4-126.

[KS06] L. Kocsis and C. Szepesvári. "Bandit Based Monte-Carlo Planning". In: *European Conference on Machine Learning (ECML 2006)*. Ed. by J. Fürnkranz, T. Scheffer, and M. Spiliopoulou. Vol. 4212. LNCS. Springer, 2006, pp. 282–293. DOI: 10.1007/11871842_29. URL: http://dx.doi.org/10.1007/11871842_29.

[Kus71] H. Kushner. *Introduction to Stochastic Control*. Holt, Rinehart and Winston, 1971.

[LH18] J. Leike and M. Heizmann. "Geometric Nontermination Arguments". In: *Tools and Algorithms for the Construction and Analysis of Systems - 24th International Conference, TACAS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings, Part II*. 2018, pp. 266–283.

[LKPK18] J. Lee, G.-H. Kim, P. Poupart, and K.-E. Kim. "Monte-Carlo Tree Search for Constrained POMDPs." In: *Neural Information Processing Systems (NeurIPS 2018)*. 2018, pp. 7934–7943.

[LMA98] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. "Planning and acting in partially observable stochastic domains". In: *Artificial intelligence* 101.1 (1998), pp. 99–134.

[LNORR14] D. Larraz, K. Nimkar, A. Oliveras, E. Rodríguez-Carbonell, and A. Rubio. "Proving Non-termination Using Max-SMT". In: *Computer Aided Verification: 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings*. Ed. by A. Biere and R. Bloem. Cham: Springer International Publishing, 2014, pp. 779–796. DOI: 10.1007/978-3-319-08867-9_52. URL: http://dx.doi.org/10.1007/978-3-319-08867-9_52.

[LPS81] D. Lehmann, A. Pnueli, and J. Stavi. "Impartiality, Justice and Fairness: The Ethics of Concurrent Termination". In: *Proceedings if ICALP'81*. Ed. by S. Even and O. Kariv. Berlin, Heidelberg: Springer Berlin Heidelberg, 1981, pp. 264–277.

[Mai13] O.-A. Maillard. "Robust Risk-Averse Stochastic Multi-Armed Bandits". In: *Algorithmic Learning Theory (ALT 2013)*. Ed. by S. Jain, R. Munos, F. Stephan, and T. Zeugmann. Berlin, Heidelberg: Springer, 2013, pp. 218–233.

[Man05] R. Mansuy. "The Origins of the Word "Martingale"". In: *Electronic Journal for History of Probability and Statistics* 5.1 (2005).

[Met87] N. C. Metropolis. "The Beginning of the Monte Carlo Method". In: *Los Alamos Science* 15 (1987), pp. 125–130.

[MG07] J. Matoušek and B. Gärtner. *Understanding and Using Linear Programming*. Springer, 2007.

[MJ84] F. L. Morris and C. B. Jones. "An Early Program Proof by Alan Turing". In: *Annals of the History of Computing* 6.2 (1984), pp. 139–143. DOI: 10.1109/MAHC.1984.10017.

[MMKK18] A. McIver, C. Morgan, B. L. Kaminski, and J. Katoen. "A New Proof Rule for Almost-Sure Termination". In: *Proceedings of the ACM on Programming Languages* 2.Proceedings of (POPL 2018) (2018), 33:1–33:28. DOI: 10.1145/3158121. URL: https://doi.org/10.1145/3158121.

[MT09] S. Meyn and R. Tweedie. *Markov Chains and Stochastic Stability*. 2009.

[MVV18] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC press, 2018.

[NSK09] M. Neuhäußer, M. Stoelinga, and J.-P. Katoen. "Delayed Nondeterminism in Continuous-Time Markov Decision Processes". In: *Proceedings of FoSSaCS 2009*. Vol. 5504. 2009, pp. 364–379.

[Pet22] J. Petrák. "Black-Box Hyperparameter Tuning for Risk-Constrained Reinforcement Learning Algorithm". MA thesis. Masaryk University, 2022.

[PJP04] S. Prajna, A. Jadbabaie, and G. Pappas. "Stochastic Safety Verification Using Barrier Certificates". In: *2004 43rd IEEE Conference on Decision and Control (CDC)*. Vol. 1. IEEE, 2004, pp. 929–934. DOI: 10.1109/CDC.2004.1428804.

[Pou+15] P. Poupart, A. Malhotra, P. Pei, K. Kim, B. Goh, and M. Bowling. "Approximate Linear Programming for Constrained Partially Observable Markov Decision Processes". In: *AAAI 2015*. AAAI Press, 2015, pp. 3342–3348. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9742.

[PR04a] A. Podelski and A. Rybalchenko. "A Complete Method for the Synthesis of Linear Ranking Functions". In: *Verification, Model Checking, and Abstract Interpretation, 5th International Conference, VMCAI 2004, Venice, January 11-13, 2004, Proceedings*. Ed. by B. Steffen and G. Levi. Springer, 2004, pp. 239–251. DOI: 10.1007/978-3-540-24622-0_20.

[PR04b] A. Podelski and A. Rybalchenko. "Transition Invariants". In: *Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science*. LICS '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 32–41. DOI: 10.1109/LICS.2004.50. URL: http://dx.doi.org/10.1109/LICS.2004.50.

[PS12] M. Petrik and D. Subramanian. "An Approximate Solution Method for Large Risk-Averse Markov Decision Processes". In: *CoRR* abs/1210.4901 (2012).

[PT87] C. H. Papadimitriou and J. N. Tsitsiklis. "The Complexity of Markov Decision Processes". In: *Mathematics of Operations Research* 12.3 (1987), pp. 441–450. DOI: 10.1287/moor.12.3.441. eprint: https://doi.org/10.1287/moor.12.3.441. URL: https://doi.org/10.1287/moor.12.3.441.

[Put05] M. L. Puterman. *Markov Decision Processes*. Wiley-Interscience, 2005, p. 684.

[Put94] M. Puterman. *Markov Decision Processes*. 1994.

[Ros06] J. S. Rosenthal. *A First Look at Rigorous Probability Theory*. 2nd. World Scientific Publishing Company, 2006.

[RRS15] M. Randour, J.-F. Raskin, and O. Sankur. "Variations on the Stochastic Shortest Path Problem". In: *Proceedings of VMCAI'15*. Ed. by D. D'Souza, A. Lal, and K. G. Larsen. Vol. 8931. LNCS. Springer, 2015, pp. 1–18. DOI: 10.1007/978-3-662-46081-8_1. URL: http://dx.doi.org/10.1007/978-3-662-46081-8_1.

[Sam+14] A. Sampson, P. Panchekha, T. Mytkowicz, K. S. McKinley, D. Grossman, and L. Ceze. "Expressing and verifying probabilistic assertions". In: *ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '14, Edinburgh, United Kingdom - June 09 - 11, 2014*. Ed. by M. F. P. O'Boyle and K. Pingali. ACM, 2014, p. 14. DOI: 10.1145/2594291.2594294. URL: http://doi.acm.org/10.1145/2594291.2594294.

[SB18] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018.

[SCG13] S. Sankaranarayanan, A. Chakarov, and S. Gulwani. "Static Analysis for Probabilistic Programs: Inferring Whole Program Properties from Finitely Many Paths". In: *PLDI*. 2013, pp. 447–458.

[SEY13] A. Stewart, K. Etessami, and M. Yannakakis. "Upper Bounds for Newton's Method on Monotone Polynomial Systems, and P-Time Model Checking of Probabilistic One-Counter Automata". In: *Proceedings of CAV 2013*. Vol. 8044. 2013, pp. 495–510.

[Sil+18] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis. "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play". In: *Science* 362.6419 (2018), pp. 1140–1144. DOI: 10.1126/science.aar6404. eprint: https://science.sciencemag.org/content/362/6419/1140.full.pdf. URL: https://science.sciencemag.org/content/362/6419/1140.

[SLM12] A. Sani, A. Lazaric, and R. Munos. "Risk-Aversion in Multi-Armed Bandits". In: *Advances in Neural Information Processing Systems (NIPS 2012)*. Vol. 25. 2012, pp. 3275–3283.

[ST12] J. Steinhardt and R. Tedrake. "Finite-Time Regional Verification of Stochastic Non-Linear Systems". In: *The International Journal of Robotics Research* 31.7 (2012), pp. 901–923. DOI: 10.1177/0278364912444146. eprint: https://doi.org/10.1177/0278364912444146. URL: https://doi.org/10.1177/0278364912444146.

[STW16] P. Santana, S. Thiébaux, and B. C. Williams. "RAO*: An Algorithm for Chance-Constrained POMDP's". In: *AAAI Conference on Artificial Intelligence (AAAI 2016)*. AAAI Press, 2016, pp. 3308–3314.

[SV10] D. Silver and J. Veness. "Monte-Carlo Planning in Large POMDPs". In: *Neural Information Processing (NIPS 23)*. Curran Associates, Inc., 2010, pp. 2164–2172. URL: http://papers.nips.cc/paper/4031-monte-carlo-planning-in-large-pomdps.pdf.

[TA11] I. Tkachev and A. Abate. "On Infinite-Horizon Probabilistic Properties and Stochastic Bisimulation Functions". In: *50th IEEE Conference on Decision and Control and European Control Conference (CDC 2011)*. IEEE, 2011, pp. 526–531. DOI: 10.1109/CDC.2011.6160617.

[TBF05] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[TGY20] D. Troop, F. Godin, and J. Y. Yu. "Risk-Averse Action Selection Using Extreme Value Theory Estimates of the CVaR". In: *CoRR* abs/1912.01718 (2020).

[Tur38] A. M. Turing. "On Computable Numbers, with an Application to the Entscheidungsproblem. A Correction". In: *Proceedings of the London Mathematical Society* 2.1 (1938), pp. 544–546.

[Tur49] A. Turing. "Checking a Large Routine". In: *Report of a Conference on High Speed Automatic Calculating Machines*. 1949, pp. 67–69.

[UH10] A. Undurti and J. P. How. "An Online Algorithm for Constrained POMDPs". In: *International Conference on Robotics and Automation (ICRA'17)*. IEEE. 2010, pp. 3966–3973.

[VR08] H. Velroyen and P. Rümmer. "Non-termination Checking for Imperative Programs". In: *Tests and Proofs: Second International Conference, TAP 2008, Prato, Italy, April 9-11, 2008. Proceedings*. Ed. by B. Beckert and R. Hähnle. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 154–170. DOI: 10.1007/978-3-540-79124-9_11. URL: http://dx.doi.org/10.1007/978-3-540-79124-9_11.

[Wil91] D. Williams. *Probability with Martingales*. Cambridge Mathematical Textbooks. Cambridge, UK: Cambridge University Press, 1991, p. 251.

[ZP96] U. Zwick and M. Paterson. "The Complexity of Mean Payoff Games on Graphs". In: *Theoretical Computer Science* 158.1–2 (1996), pp. 343–359.

Part II

COLLECTION OF APPLICANT'S PUBLICATIONS

LIST OF ENCLOSED PUBLICATIONS

7

Below we list all the papers submitted as a part of this habilitation thesis. We note that they do not represent the applicant's entire bibliography. The rules of the habilitation procedure at Masaryk University require us to provide an estimate of the author's percentual contribution towards each of the papers. Since such a percentage is impossible to estimate correctly in practice, we use 100%/number of authors as a proxy measure. The author's factual contribution is sketched under each record.

1. K. Chatterjee, H. Fu, P. Novotný, and R. Hasheminezhad. "Algorithmic Analysis of Qualitative and Quantitative Termination Problems for Affine Probabilistic Programs". In: *ACM Trans. Program. Lang. Syst.* 40.2 (2018), 7:1–7:45. DOI: 10.1145/3174800. URL: https://doi.org/10.1145/3174800 25%

   - Contributed equally to all parts of the research and writing process.

2. K. Chatterjee, P. Novotný, and D. Žikelić. "Stochastic invariants for probabilistic termination". In: *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL 2017)*. Ed. by G. Castagna and A. D. Gordon. ACM, 2017, pp. 145–160. DOI: 10.1145/3009837. URL: http://dl.acm.org/citation.cfm?id=3009873 33.3%

   - Originator of the main idea of the paper, proved most of the key results, wrote a large majority of the resulting text, overseen the prototype implementation.

3. S. Agrawal, K. Chatterjee, and P. Novotný. "Lexicographic ranking supermartingales: an efficient approach to termination of probabilistic programs". In: *Proc. ACM Program. Lang.* 2. (Proceedings of POPL'18) (2018), 34:1–34:32. DOI: 10.1145/3158122. URL: https://doi.org/10.1145/3158122 33.3%

   - Originator of the main idea of the paper, proved most of the key results, wrote a large majority of the resulting text, overseen the prototype implementation.

4. K. Chatterjee, E. K. Goharshady, P. Novotný, J. Zárevúcky, and D. Žikelić. "On Lexicographic Proof Rules for Probabilistic Termination". In: *Proceedings of FM'21*. Ed. by M. Huisman, C. S. Pasareanu, and N. Zhan. Vol. 13047. LNCS. Springer, 2021, pp. 619–639 20%

   - Originator of the main idea of the paper, wrote approximately half of the paper.

5. K. Chatterjee, H. Fu, and P. Novotný. "Termination Analysis of Probabilistic Programs with martingales". In: *Foundations of Probabilistic Programming*. Ed. by G. Barthe, J.-P. Katoen, and A. Silva. Cambridge University Press, 2020. Chap. 7, pp. 221–258 33.3%

    - Contributed equally to all parts of the research and writing process.

6. K. Chatterjee, E. K. Goharshady, P. Novotný, and D. Žikelić. "Proving Non-Termination by Program Reversal". In: *Proceedings of PLDI'21*. Ed. by S. N. Freund and E. Yahav. ACM, 2021, pp. 1033–1048 25%

    - Significantly contributed to the conceptual discussions behind the paper, equally contributed to the writing of the paper.

7. K. Chatterjee, P. Novotný, G. A. Pérez, J. Raskin, and D. Žikelić. "Optimizing Expectation with Guarantees in POMDPs". In: *Proceedings of AAAI'17*. AAAI Press, 2017, pp. 3725–3732. URL: http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14354 20%

    - Contributed equally to all parts of the research and writing process.

8. K. Chatterjee, A. Elgyütt, P. Novotný, and O. Rouillé. "Expectation Optimization with Probabilistic Guarantees in POMDPs with Discounted-Sum Objectives". In: *Proceedings of IJCAI'18*. Ed. by J. Lang. ijcai.org, 2018, pp. 4692–4699. DOI: 10.24963/ijcai.2018/652. URL: https://doi.org/10.24963/ijcai.2018/652 25%

    - Originator of the main idea of the paper, wrote a large majority of the resulting text, overseen the prototype implementation.

9. T. Brázdil, K. Chatterjee, P. Novotný, and J. Vahala. "Reinforcement Learning of Risk-Constrained Policies in Markov Decision Processes". In: *The Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI 2020)*. AAAI Press, 2020, pp. 9794–9801. URL: https://aaai.org/ojs/index.php/AAAI/article/view/6531 25%

    - Originator of the main idea of the paper, wrote a large majority of the resulting text, overseen the prototype implementation.

The pages containing the enclosed papers were removed from the public version to prevent copyright infringement.