

Automatisierung der Terminierungsanalyse von Probabilistischen Programmen

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Logic and Computation

eingereicht von

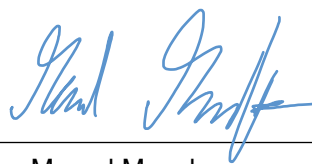
Marcel Moosbrugger, BSc

Matrikelnummer 01426526

an der Fakultät für Informatik
der Technischen Universität Wien

Betreuung: Univ.Prof. Dr.techn. Laura Kovács, MSc

Wien, 22. Mai 2020



Marcel Moosbrugger



Laura Kovács



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Automating Termination Analysis of Probabilistic Programs

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Logic and Computation

by

Marcel Moosbrugger, BSc

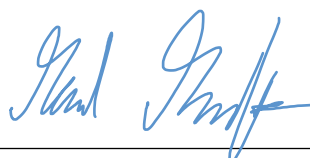
Registration Number 01426526

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dr.techn. Laura Kovács, MSc

Vienna, 22nd May, 2020



Marcel Moosbrugger



Laura Kovács



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Erklärung zur Verfassung der Arbeit

Marcel Moosbrugger, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 22. Mai 2020



Marcel Moosbrugger



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Acknowledgements

I want to especially thank my advisor Laura Kovács as well as my co-advisors Ezio Bartocci and Joost-Pieter Katoen for the inspiring discussions over the last months. The constant detailed feedback from Laura Kovács had a tremendous impact on the quality of this thesis.

The Bachelor with Honors program at the Faculty of Informatics of TU Wien allowed me to immerse myself in the beauty of academic research and the pursuit of knowledge early on. I am thankful to Ulrich Schmid for initiating and developing the program with great enthusiasm as well as to my mentor Thomas Eiter for his guidance.

The research grants ERC Starting Grant SYMCAR 639270 and the ERC Proof of Concept Grant SYMELS 842066, partially supported activities that led to this thesis.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Kurzfassung

Die Terminierung von Computerprogrammen zu entscheiden ist eines der ersten und berüchtigtsten Probleme der Informatik. In dieser Arbeit automatisieren wir die Terminierungsanalyse einer Klasse von probabilistischen Programmen, der Klasse sogenannter Prob-solvable loops, mithilfe von Beweisregeln basierend auf Supermartingalen. Wir konstruieren Algorithmen für almost-sure-termination, positive-almost-sure-termination, sowie für die Negationen der Konzepte. Für diesen Zweck nutzen wir strukturelle Eigenschaften von Prob-solvable loops. Die Eigenschaften ermöglichen uns asymptotische Schranken für polynomielle Ausdrücke über Programmvariablen automatisch zu berechnen. Diese Schranken werden dann benutzt, um die Bedingungen der probabilistischen Beweisregeln, wie zum Beispiel die Bedingung für Supermartingale, zu überprüfen.

Um die Negation von almost-sure-termination festzustellen, verallgemeinern wir existierende Beweisregeln, die auf repulsiven Supermartingalen basieren. Dies ermöglicht uns unbeschränkte, polynomielle Updates von Programmvariablen zu unterstützen. Die verallgemeinerte Beweisregel ist für allgemeine probabilistische Programme verwendbar und nicht nur für Prob-solvable loops. Weiters, identifizieren wir eine Subklasse von probabilistischen Programmen, für die wir einen vollständigen und korrekten Algorithmus entwickeln, welcher almost-sure-termination für Programme der Subklasse entscheidet. Unsere identifizierte Subklasse ist strikt größer als die Klasse sogenannter constant-probability-programs, welches die größte derzeit bekannte Klasse ist, für die almost-sure-termination entscheidbar ist.

Wir kombinieren die entwickelten Algorithmen für die probabilistische Terminierungsanalyse in unserem neuen Softwaretool AMBER. Experimentelle Ergebnisse zeigen, dass AMBER probabilistische Programme handhaben kann, die unerreichbar für andere state-of-the-art Tools sind.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Abstract

Deciding termination of computer programs is one of the most infamous challenges in computer science. In this thesis, we automate the termination analysis of a class of probabilistic programs, called Prob-solvable loops, through (super-)martingale based proof rules. We establish incomplete but sound algorithms for almost-sure termination, positive-almost-sure termination, and the negations thereof. We achieve this, by exploiting the structural restrictions of Prob-solvable loops. The restrictions let us effectively compute asymptotic bounds on polynomial expressions of program variables. These bounds are then used to decide the preconditions of the probabilistic termination proof rules, like the supermartingale condition.

For certifying the negation of almost-sure termination, we generalize existing proof rules involving repulsing supermartingales, to handle unbounded polynomial variable updates of programs. This generalization applies to general probabilistic programs even beyond Prob-solvable loops. Moreover, we identify a subclass of probabilistic programs and introduce a sound and complete procedure deciding almost-sure termination of such programs. Our identified subclass strictly subsumes the class of so-called constant probability programs, the largest decidable subclass currently known.

We combine our proposed algorithms for probabilistic termination analysis in our new tool AMBER. Experimental results demonstrate that AMBER can handle probabilistic programs that are out of reach for other state-of-the-art tools.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Contents

Kurzfassung	ix
Abstract	xi
Contents	xiii
1 Introduction	1
1.1 Motivation & Problem Statement	1
1.2 Contributions	2
1.3 Methodology	3
1.4 Structure of the Thesis	3
2 Preliminaries	5
2.1 Probabilistic Programs (PPs)	5
2.2 Prob-solvable Loops	6
2.3 Canonical Probability Space	7
2.4 Probabilistic Termination	9
2.5 Martingale Theory	10
3 Proof Rules Certifying Termination Properties	15
3.1 Positive-Almost-Sure-Termination (PAST)	15
3.2 Almost-Sure-Termination (AST)	17
3.3 Non-Termination	18
3.4 Polynomial Non-Termination	20
3.5 Relaxations of Proof Rules	26
3.6 Completeness	29
4 Automating Termination Analysis of Probabilistic Programs	35
4.1 Prob-solvable Loops and Monomials	39
4.2 Asymptotic Bounds for Prob-solvable Loops	41
4.3 Algorithms for Proof Rules	47
4.4 Rule out Rules	51
4.5 Implementation and Evaluation	53
	xiii

5	Related Work	59
6	Conclusion	61
	Acronyms	63
	Bibliography	65

Introduction

1.1 Motivation & Problem Statement

The problem of deciding the termination behavior of traditional programs, first considered by Alan Turing and known as the *Halting Problem* [Tur37], essentially initiated *Computer Science* as well as *Computability Theory* and emerged from the field of *Mathematical Logic*. Since first stated by Turing, it is well known that the *Halting Problem* is undecidable. However, for which non-trivial subclasses of all possible instances, an effective procedure deciding termination can be given, was heavily researched in the second half of the twentieth century and still is to this day [CPR06], in the rise of new programming languages and principles.

Probabilistic programming is one such emerging paradigm [Gha15]. In a nutshell, probabilistic programming allows for specifying probabilistic models (e.g. Bayesian networks, Markov decision processes) as programs in a suitable language and provides automatic inference for the specified models. These programs are referred to as probabilistic programs (PPs). In the wider sense, also randomized algorithms can be thought of as PPs. In [Gha15], Zoubin Ghahramani wrote that there are several reasons why probabilistic programming could prove to be *revolutionary* for machine intelligence and scientific modeling, such as Bayesian optimization, probabilistic data compression or automating the discovery of plausible and interpretable models from data.

A natural question that arises in PPs is one of the most central problems in computer science: How can PPs be proven terminating? What is the termination behavior of PPs? For which subclasses of PPs can we effectively answer this question? Since PPs are employed in safety-critical environments [ARS13] [BBR⁺15] [FDG⁺19], providing formal guarantees for these programs is not only a natural challenge but a necessary task to solve.

In this thesis we will study classes of PPs, in particular, the restricted class of *Prob-solvable loops* [BKS19], with the overall goal of automating the termination analysis of such programs.

1.2 Contributions

The contributions of this thesis are abundant. We provide the following results to improve the state-of-the-art:

1. Proof Rules: A novel proof rule for certifying non-termination of PPs with polynomial variable updates (Chapter 3, Section 3.4),
2. Complete Subclass: A characterization of a subclass of PPs for which the termination rules are complete (Chapter 3, Section 3.6),
3. Automation: Methods enabling automation of selected proof rules together with and implementations thereof in our tool AMBER (Chapter 4).

1.2.1 Proof Rules

In the last few years, quite a few proof rules involving martingales, for example [CS13] [FFH15] [MMKK17] [CNZ17] [ACN17], have been proposed (see Chapter 3). Applied to a PP, they reveal something about its termination behavior. The proof rules for non-termination (i.e. there is a non-zero probability of not terminating) come with restricting assumptions, rendering them useless for PPs beyond constant variable updates. *We propose a new proof rule for non-termination applicable to a wider range of PPs than the current state-of-the-art allows for.*

1.2.2 Complete Subclass

Faced with an individual PP, a termination proof rule can be applied or not, depending on the preconditions of the proof rule and the program. The termination behavior of PPs containing just constant updates is known to allow for a simple decision procedure [GGH19]. Testing membership of the identified subclass should, of course, be smaller in complexity than deciding termination itself. *In this master thesis, we characterize a class of PPs strictly subsuming the class of constant update programs and provide a decision procedure deciding its termination behavior.*

1.2.3 Automation

Implementing several state-of-the-art termination proof rules for PPs has not yet been done. *In this thesis, we propose our new tool AMBER, implementing several proof rules for termination as well as non-termination.* We extend the class of Prob-solvable programs [BKS19] by adding loop guards and implement the proof rules for this class of programs.

Moreover, automating the mentioned proof rules is not merely an implementation task but also requires lemmas and theorems providing shortcuts to the preconditions of the rules. *Developing these lemmas and theorems is also part of this thesis.*

1.3 Methodology

We establish the results of this thesis using the following methodologies:

- Extending the state-of-the-art of probabilistic termination proof rules is done through *deduction*. We give mathematical definitions of PPs and other required notions. The new proof rules are stated in the form of theorems and justified with deductive arguments. We make heavy use of the well-known mathematical theory of martingales to state and prove our results.
- Deduction is also the method of choice for identifying the subclass of probabilistic programs for which the proof rules are complete.
- AMBER is evaluated empirically. We compare our tool against other existing solutions, wherever available.

1.4 Structure of the Thesis

In Chapter 2 all notions on which the rest of the thesis builds are introduced. Concepts from the area of *programming languages* as well as from *probability theory* are given. Moreover, the chapter formalizes our research questions on the termination of PPs.

Subsequently, in Chapter 3 we state already existing proof rules for different probabilistic termination properties. Furthermore, we introduce our new proof rule certifying non-termination for probabilistic programs and characterize a fragment of programs for which the presented proof rules constitute a complete decision procedure.

Chapter 4 automates the previously established proof rules for the fragment of Prob-solvable loops. We provide sound relaxations of the mentioned proof rules enabling us to decide the probabilistic termination properties from Chapter 2 automatically.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Preliminaries

2.1 Probabilistic Programs (PPs)

The syntax of PPs is usually defined in the form of a while-language [MM06] [CS13] [CNZ17] [HKGK19]. Additionally to the typical language constructs present in a while-language, PPs are equipped with a probabilistic branching mechanism $branch_1 [p] branch_2$. Intuitively, $branch_1 [p] branch_2$ is interpreted as the program run of the program $branch_1$ with probability p and with probability $1 - p$ as the program run of $branch_2$.

Operationally, a PP constitutes a *Markov Decision Process (MDP)* [GKM12]. A PP can be semantically described as a probabilistic transition system [CS13] or as a probabilistic control flow graph [CNZ17] which in turn induce an MDP together with its associated probability space.

In what follows, we introduce a class of PPs called *Prob-solvable loops* [BKS19]. Prob-solvable loops are a restriction of general PPs expressible in a while-language as defined for example in [MM06]. The major advantage of Prob-solvable loops is that their program variables can be represented in an algebraically closed form only depending on the loop counter. Moreover, these closed forms can be calculated automatically from recurrence relations. This fact is heavily exploited when automating termination analysis for Prob-solvable loops.

Afterwards, we will directly define the probability space represented by a Prob-solvable loop. This process is not specific to Prob-solvable loops and can be analogously applied to general PPs. Along the way, we will state some general notions and results from probability theory. For a more detailed introduction to probability theory and MDPs, we refer to [Wil91], [GGG⁺01], [KSK76].

2.2 Prob-solvable Loops

We will slightly modify the notion of *Prob-solvable loops* introduced in [BKS19] to be able to study their termination behavior. In [BKS19], the authors were not concerned with termination and only allowed the trivial loop guard *true*. Therefore, we extend the original definition with non-trivial loop guards. Moreover, we restrict all coefficients to be from \mathbb{R} and do not allow probability distributions as coefficients. In what follows, we refer with *Prob-solvable loop* to the following programs.

Definition 1 (Syntax of Prob-solvable loops). *A Prob-solvable loop with variables x_1, \dots, x_m , where $m \in \mathbb{N}$, is a program of the form*

$$\mathcal{I} \text{ while } \mathcal{G} \text{ do } \mathcal{U} \text{ end}$$

such that

- \mathcal{I} (initialization) is a sequence of m assignments $x_1 := r_1, \dots, x_m := r_m$ for $r_i \in \mathbb{R}$;
- \mathcal{G} (guard) is a strict inequality $P > Q$, where P and Q are polynomials over the program variables;
- \mathcal{U} (update) is a sequence of m probabilistic updates of the form $x_i := a_{i1}x_i + P_{i1} [p_{i1}] \ a_{i2}x_i + P_{i2} [p_{i2}] \ \dots \ [p_{il_i}] \ a_{i(l_i+1)}x_i + P_{i(l_i+1)}$, where

$a_{ij} \in \mathbb{R}$ are real constants and $P_{ij} \in R[x_1, \dots, x_{i-1}]$ are polynomials. Further, for every i it has to hold that $\sum_j p_{ij} < 1$ and $p_{ij} \in [0, 1]$, for all j .

For a Prob-solvable loop \mathcal{L} we refer to its initialization, guard and update parts by $\mathcal{I}_{\mathcal{L}}$, $\mathcal{G}_{\mathcal{L}}$ and $\mathcal{U}_{\mathcal{L}}$ respectively. If the loop \mathcal{L} is clear from context, the subscript will be omitted. Throughout a Prob-solvable loop, symbolic constants can be used instead of concrete real numbers.

Intuitively, an update like $x := x + 2 [p] \ x - 3$ means that the program variable x gets incremented by 2 with probability p and decremented by 3 with probability $1 - p$. The semantics of Prob-solvable loops will be precisely stated in the next section by associating a Prob-solvable loop with a probability space.

A notable property of Prob-solvable loops is that the statistical moments of program variables can be expressed in closed-forms, that means just dependent on the loop counter [BKS19]. Moreover, these closed-forms always exist. The main restriction enabling this is that a program variable x is, figuratively speaking, only allowed to depend on other variables preceding x in the loop body.

An example of a Prob-solvable loop is the following program which implements the well-known symmetric 2D-random-walk.

Example 1 (Symmetric 2D-random-walk).

```

left := 0
right := 0
top := 0
down := 0
x := x0
y := y0
while x2 + y2 > 0 do
  left := 1 [1/4] 0
  right := 1 - left [1/3] 0
  top := 1 - (left + right) [1/2] 0
  down := 1 - (left + right + top)
  x := x + right - left
  y := y + top - down
end

```

2.3 Canonical Probability Space

In order to define a canonical probability space for a given Prob-solvable loop, we fix a Prob-solvable loop \mathcal{L} for the remainder of this chapter. As already mentioned, a PP operationally corresponds to an MDP. An MDP induces a special probability space, known in the literature as its *sequence space* [KSK76]. We will associate to the loop \mathcal{L} the sequence space of its corresponding MDP. Our approach closely follows the one in [HKGK19] where the authors also associate a PP with the sequence space of its corresponding MDP. We begin by defining the notion of a *state* and a *run* for a Prob-solvable loop.

Definition 2 (State). *The state of the Prob-solvable loop \mathcal{L} after a given number of loop iterations $i \in \mathbb{N}$ is a vector $s \in \mathbb{R}^m$ where m is the number variables in \mathcal{L} . With $s[i]$ or $s[x_i]$ we will denote the i -th component of the vector s representing the value of the variable x_i in state s .*

Definition 3 (Run). *A run ϑ of \mathcal{L} is an infinite sequence of states. So $\vartheta = s_0s_1s_2s_3\dots$ where every s_i is a state.*

Note that any infinite sequence of states classifies as a *run*. Later on, however, infeasible runs will be given measure 0.

A probability space $(\Omega, \Sigma, \mathbb{P})$ consists of a measurable space (Ω, Σ) and a probability measure \mathbb{P} for this space. Let us first define a measurable space for \mathcal{L} and then equip it with a probability measure.

Definition 4 (Loop Space). *The Prob-solvable loop \mathcal{L} induces a canonical measurable space $(\Omega^{\mathcal{L}}, \Sigma^{\mathcal{L}})$ where the sample space $\Omega^{\mathcal{L}} := (\mathbb{R}^m)^\omega$, that means $\Omega^{\mathcal{L}}$ is the set of all infinite sequences of program states or in other words the set of all program runs.*

$\Sigma^{\mathcal{L}}$ is the smallest σ -field containing all cylinder sets $Cyl(\pi) := \{\pi\vartheta \mid \vartheta \in (\mathbb{R}^m)^\omega\}$ for all finite prefixes $\pi \in (\mathbb{R}^m)^+$, that is

$$\Sigma^{\mathcal{L}} = \langle \{Cyl(\pi) \mid \pi \in (\mathbb{R}^m)^+\} \rangle_\sigma.$$

To complete the loop space to a proper probability space for which probabilistic questions can be stated, we need to define a probability measure. We continue by defining the probability $p(\pi)$ of a finite non-empty prefix π of a program run.

$$p(s) = \mu_{\mathcal{I}}(s) \tag{2.1}$$

$$p(\pi s s') = \begin{cases} p(\pi s) \cdot [s' = s], & \text{if } s \models \neg \mathcal{G} \\ p(\pi s) \cdot \mu_{\mathcal{U}}(s, s'), & \text{if } s \models \mathcal{G} \end{cases} \tag{2.2}$$

In the definition above, $\mu_{\mathcal{I}}(s)$ denotes the probability that after initialization the loop \mathcal{L} is in state s . Because probabilistic constructs are not allowed in \mathcal{I} , $\mu_{\mathcal{I}}(s)$ is a Delta-distribution such that $\mu_{\mathcal{I}}(s) = 1$ for the unique state s defined by \mathcal{I} and $\mu_{\mathcal{I}}(s') = 0$ for $s' \neq s$. Moreover, $\mu_{\mathcal{U}}(s, s')$ denotes the probability that after one iteration of the loop body \mathcal{U} starting from state s the resulting program state is s' . These probabilities are, as notation suggests, solely determined by the initialization \mathcal{I} and the loop body \mathcal{U} respectively.

Intuitively, for a finite non-empty prefix π of a program run, the value $p(\pi)$ is the probability that when running \mathcal{L} , that π is the sequence of the first $|\pi|$ program states. Note that we are taking a loop-based approach where the effect of the initialization and the effect of the loop-body are captured in a single states.

Now we have everything at hand for defining the canonical probability measure for the loop space.

Definition 5 (Loop Measure). *For the Prob-solvable loop \mathcal{L} a canonical probability measure, called Loop Measure, on its loop space is given by $\mathbb{P}^{\mathcal{L}} : \Sigma^{\mathcal{L}} \rightarrow [0, 1]$ with*

$$\mathbb{P}^{\mathcal{L}}(Cyl(\pi)) = p(\pi)$$

The loop space together with the loop measure forms a probability space, i.e. $(\Omega^{\mathcal{L}}, \Sigma^{\mathcal{L}}, \mathbb{P}^{\mathcal{L}})$ is a probability space. Whenever the Prob-solvable loop is clear from context, the superscript \mathcal{L} will be omitted.

2.4 Probabilistic Termination

Let $\overline{\mathbb{R}}$ denote $\mathbb{R} \cup \{+\infty, -\infty\}$. Random variables X will be of central interest and are as usual in probability theory defined as (Σ -)measurable functions $X : \Omega \rightarrow \overline{\mathbb{R}}$ for a probability space $(\Omega, \Sigma, \mathbb{P})$. As a reminder, X being measurable means that for every open set $U \subseteq \overline{\mathbb{R}}$ it holds that $X^{-1}(U) \in \Sigma$. The expected value of X , denoted by $\mathbb{E}(X)$, is defined as the Lebesgue integral of X over the whole space i.e. $\mathbb{E}(X) := \int_{\Omega} X d\mathbb{P}$. In the special case that X takes only countably many values we get:

$$\mathbb{E}(X) = \int_{\Omega} X d\mathbb{P} = \sum_{r \in X(\Omega)} \mathbb{P}(X = r) \cdot r$$

An essential random variable is the *Looping time* which allows us to formalize different termination properties for Prob-solvable loops.

Definition 6 (Looping Time). *The random variable*

$$T^{-\mathcal{G}} : \Omega \rightarrow \mathbb{N} \cup \{\infty\}, \vartheta \mapsto \inf\{n \in \mathbb{N} \mid \vartheta[n] \models \neg \mathcal{G}\}$$

is called the looping time of \mathcal{L} .

Intuitively, the looping time maps a given program run to the index of the first state falsifying the loop guard or to ∞ if no such state exists. With this notion at hand we can finally formalize different termination properties.

Definition 7 (Termination). *The Prob-solvable loop \mathcal{L} is defined to be almost-surely-terminating (AST) if $\mathbb{P}(T^{-\mathcal{G}} < \infty) = 1$. If additionally $\mathbb{E}(T^{-\mathcal{G}}) < \infty$ holds, then \mathcal{L} is said to be positively-almost-surely-terminating (PAST).*

For traditional deterministic programs the notions AST and PAST coincide. If a deterministic program terminates with probability 1, it does so in a finite expected time. However, for a probabilistic program it is possible to reach a terminating state with probability 1 and on average require an infinite amount of time. It is a well-known mathematical result that this is the case for the previously introduced symmetric 2D-random-walk. A smaller example which is AST but not PAST is the symmetric 1D-random-walk. Similar as for the symmetric 2D-random-walk, also the symmetric 1D-random-walk can be encoded as a Prob-solvable loop.

Example 2 (Symmetric 1D-random-walk).

```

x := x0

while x > 0 do
  | x := x + 1 [1/2] x - 1
end

```

The symmetric 1D-random walk is a random walk on the number line. Starting at some positive number x_0 , x "walks" to the left or to the right with equal probability of $\frac{1}{2}$. x reaches the point 0 with probability 1. However, the expected time for this event to occur is ∞ . Hence, the symmetric 1D-random walk is not PAST, but is AST.

2.5 Martingale Theory

In some cases, $\mathbb{P}(T^{-\mathcal{G}} < \infty)$ and $\mathbb{E}(T^{-\mathcal{G}} < \infty)$ can be effectively computed. For general PPs, however, these quantities are uncomputable. The reason for that is that classical deterministic programs are special PPs and the ability to compute either $\mathbb{P}(T^{-\mathcal{G}} < \infty)$ or $\mathbb{E}(T^{-\mathcal{G}} < \infty)$ would imply the ability to solve the Halting Problem, which is undecidable.

As a result, sufficient conditions for AST, PAST and their negations have been developed (see Chapter 3). Most of these sufficient conditions make use of some notion related to (super-)martingales. Martingales are special stochastic processes.

Definition 8 (Stochastic Process). A stochastic process $(X_i)_{i \in \mathbb{N}}$ is a sequence of random variables. Every arithmetic expression E over the program variables of the Prob-solvable loop \mathcal{L} induces a stochastic process $(E_i)_{i \in \mathbb{N}}$:

$$E_i : \Omega \rightarrow \mathbb{R} \quad E_i(\vartheta) := E(\vartheta_i) \quad (2.3)$$

That means, for a given program run ϑ , $E_i(\vartheta)$ is the evaluation of E in the i -th state of the program run.

For a boolean proposition B containing program variables, we refer by B_i to the result of substituting every program variable x by x_i in the proposition B .

Consider again the 1D-random walk as an example, the stochastic process $(x_i)_{i \in \mathbb{N}}$ is such that every x_i maps a given program run to the value of the variable x in the i -th state of the program run.

It is worth mentioning that up to this point no notion of *execution* of a PP has been introduced. All that happened is, we considered all possible sequences of states as program runs and defined a suitable probability measure ruling out impossible runs. The σ -algebra $\Sigma^{\mathcal{L}}$ contains the cylinder sets for finite program run prefixes of *arbitrary* length. In $\Sigma^{\mathcal{L}}$ we can differentiate between any two distinct program runs no matter at which point they first become different. This does not capture the gradual information gain when executing a program. After executing the initialization of the loop \mathcal{L} we only know the first state of the program run. We cannot differentiate between program runs differing only after the first state. If afterward the loop body gets executed once, we gain information about the second state of the run and so on. In probability theory, there is a well-known notion to capture the information available at a certain point in time, called *filtration*.

Definition 9 (Filtration). For a probability space $(\Omega, \Sigma, \mathbb{P})$, a filtration is a sequence $(\mathcal{F}_i)_{i \in \mathbb{N}}$ such that

- every \mathcal{F}_i is a sub- σ -algebra,
- and $\mathcal{F}_i \subseteq \mathcal{F}_{i+1}$.

$(\Omega, \Sigma, (\mathcal{F}_i)_{i \in \mathbb{N}}, \mathbb{P})$ is called a filtered probability space.

We can use this concept to define a specific filtration modeling the gradual information gain inherent in executing a program. We further enrich the loop space with the *loop filtration* to form a filtered probability space.

Definition 10 (Loop Filtration). The sequence $(\mathcal{F}_i^{\mathcal{L}})_{i \in \mathbb{N}}$ is a filtration of $\Sigma^{\mathcal{L}}$, where

$$\mathcal{F}_i^{\mathcal{L}} = \langle \{Cyl(\pi) \mid \pi \in (\mathbb{R}^m)^+, |\pi| = n + 1\} \rangle_{\sigma}.$$

$(\mathcal{F}_i^{\mathcal{L}})_{i \in \mathbb{N}}$ is called loop filtration. $(\Omega^{\mathcal{L}}, \Sigma^{\mathcal{L}}, (\mathcal{F}_i^{\mathcal{L}})_{i \in \mathbb{N}}, \mathbb{P}^{\mathcal{L}})$ forms a filtered probability space. If \mathcal{L} is clear from context, the superscript will be omitted.

$\mathcal{F}_0^{\mathcal{L}}$ is the smallest σ -algebra containing the cylinder sets of finite prefixes of program runs of length 1. Therefore, the cylinder sets of finite prefixes of program runs of length 2 and higher are not present in $\mathcal{F}_0^{\mathcal{L}}$. This means $\mathcal{F}_0^{\mathcal{L}}$ captures exactly the information available about the program run after executing just the initialization of \mathcal{L} . Similarly, $\mathcal{F}_1^{\mathcal{L}}$ captures the information about the program run after the loop body has been executed once, $\mathcal{F}_2^{\mathcal{L}}$ after two executions and so on.

Going back to the stochastic process $(x_i)_{i \in \mathbb{N}}$ of the symmetric 1D-random-walk, we observe that the event $\{\vartheta \in \Omega \mid x_i(\vartheta) = r\}$ denoted by $\{x_i = r\}$ is $\mathcal{F}_i^{\mathcal{L}}$ -measurable for every $i \in \mathbb{N}$ and every $r \in \mathbb{R}$. Intuitively, this means that the value of x_i depends only on information available up to the i -th iteration of the loop body and not on future information. This is captured in the following notion.

Definition 11 (Adapted Process). A stochastic process $(X_i)_{i \in \mathbb{N}}$ is said to be adapted to a filtration $(\mathcal{F}_i)_{i \in \mathbb{N}}$ if X_i is \mathcal{F}_i -measurable for every $i \in \mathbb{N}$.

In fact, it is easy to see that for every arithmetic expression E over the variables of the Prob-solvable loop \mathcal{L} , its stochastic process $(E_i)_{i \in \mathbb{N}}$ is adapted to the loop filtration because the value of E_i only depends on the information available up to the i -th loop iteration.

With the loop filtration at hand, we may want to condition the expected values of random variables on some $\mathcal{F}_i^{\mathcal{L}}$. Naturally speaking, we would like to ask about the expected value of a random variable X given the information about the first i loop iterations, in symbols $\mathbb{E}(X \mid \mathcal{F}_i^{\mathcal{L}})$. Formally, conditional expected values are defined as follows.

Definition 12 (Conditional Expected Value). *For a probability space $(\Omega, \Sigma, \mathbb{P})$, an integrable random variable X and a sub- σ -algebra $\Delta \subseteq \Sigma$, the expected value of X conditioned on Δ , $\mathbb{E}(X \mid \Delta)$, is any Δ -measurable function such that for every $D \in \Delta$ we have*

$$\int_D \mathbb{E}(X \mid \Delta) d\mathbb{P} = \int_D X d\mathbb{P}. \quad (2.4)$$

The random variable $\mathbb{E}(X \mid \Delta)$ is almost surely unique.

Martingales and related notions are of central importance for probabilistic termination proof rules, i.e. for sufficient conditions certifying AST, PAST or their negations. Up to this point, all required concepts for defining martingales have been introduced.

Definition 13 (Martingale). *Let $(\Omega, \Sigma, (\mathcal{F}_i)_{i \in \mathbb{N}}, \mathbb{P})$ be a filtered probability space and $(M_i)_{i \in \mathbb{N}}$ be an integrable stochastic process adapted to $(\mathcal{F}_i)_{i \in \mathbb{N}}$. Then $(M_i)_{i \in \mathbb{N}}$ is a martingale if*

$$\mathbb{E}(M_{i+1} \mid \mathcal{F}_i) = M_i. \quad (2.5)$$

Moreover, $(M_i)_{i \in \mathbb{N}}$ is called a supermartingale (SM) if

$$\mathbb{E}(M_{i+1} \mid \mathcal{F}_i) \leq M_i. \quad (2.6)$$

For an arithmetic expression E over the program variables, we refer by E 's martingale expression to the expression $\mathbb{E}(E_{i+1} - E_i \mid \mathcal{F}_i)$.

Remark 1. *Due to the linearity of the conditional expected value, the defining conditions $\mathbb{E}(M_{i+1} \mid \mathcal{F}_i) = M_i$ and $\mathbb{E}(M_{i+1} \mid \mathcal{F}_i) \leq M_i$ are equivalent to $\mathbb{E}(M_{i+1} - M_i \mid \mathcal{F}_i) = 0$ and $\mathbb{E}(M_{i+1} - M_i \mid \mathcal{F}_i) \leq 0$ respectively.*

Returning to the symmetric 1D-random-walk and its stochastic process $(x_i)_{i \in \mathbb{N}}$, we can compute $\mathbb{E}(x_{i+1} \mid \mathcal{F}_i) = \frac{1}{2}(x_i + 1) + \frac{1}{2}(x_i - 1) = x_i$, which means that x is a martingale. Because every martingale is also a SM, x is of course also a SM. A Prob-solvable loop and a stochastic process which is a SM but not a martingale is for example given by an asymmetric 1D-random-walk favoring to decrease the value of x .

Example 3 (Asymmetric 1D-random-walk). *Let $\epsilon > 0$ be fixed.*

$x := x_0$

while $x > 0$ **do**

 | $x := x + 1 \lfloor \frac{1}{2} - \epsilon \rfloor x - 1$

end

For this program we get with a simple calculation $\mathbb{E}(x_{i+1} | \mathcal{F}_i) = (\frac{1}{2} - \epsilon)(x_i + 1) + (\frac{1}{2} + \epsilon)(x_i - 1) = x_i - 2\epsilon < x_i$. Therefore, x is a SM but not a martingale.

Finally, we want to draw attention to the notion of a *stopping time*.

Definition 14 (Stopping Time). *Let $(\Omega, \Sigma, (\mathcal{F}_i)_{i \in \mathbb{N}}, \mathbb{P})$ be a filtered probability space. A random variable $T : \Omega \rightarrow \mathbb{N} \cup \{\infty\}$ is called stopping time if $\{T = i\} \in \mathcal{F}_i$ for all $i \in \mathbb{N}$.*

A stopping time models the time at which a stochastic process exhibits a specific property and should be stopped. Usually, a stopping time is defined by an underlying criterion, which describes when a stochastic process should be stopped. The defining condition $\{T = i\} \in \mathcal{F}_i$ states that this underlying criterion is only allowed to be based on the present or past and must not consider information from the future. For a stochastic process X and a stopping time T , the stopped process is denoted by X_T and defined by

$$X_{T_i}(\vartheta) := X_{\min\{i, T(\vartheta)\}}(\vartheta). \quad (2.7)$$

Considering the filtered loop space $(\Omega^{\mathcal{L}}, \Sigma^{\mathcal{L}}, (\mathcal{F}_i^{\mathcal{L}})_{i \in \mathbb{N}}, \mathbb{P}^{\mathcal{L}})$, the random variable looping time $T^{-\mathcal{G}}$ is a stopping time for this space. That is because to determine whether or not the looping time is n for a given run, it suffices to know the first $n + 1$ states of the program run (initial state plus n loop iterations).

A central theorem connecting martingales and stopping times is the *Optional Stopping Theorem*.

Theorem 1 (Optional Stopping Theorem [Wil91]). *Let T be a stopping time and M a SM. Assume T is bounded (i.e. for some $N \in \mathbb{N}$, $T(\vartheta) \leq N$ for all $\vartheta \in \Omega$) Then M_T is integrable and*

$$\mathbb{E}(M_T) \leq \mathbb{E}(M_0)$$

If M is a martingale $\mathbb{E}(M_T) = \mathbb{E}(M_0)$.

In the literature, there are different versions of this theorem with different preconditions. Most preconditions require the stopping time to be bounded in one way or another. Other works also require bounds on M . In [Wil91, 10.10] the author lists the different preconditions in a single theorem.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Proof Rules Certifying Termination Properties

The two major termination properties for probabilistic programs are *almost-sure-termination* (*AST*) and *positive-almost-sure-termination* (*PAST*). As explained in Chapter 2, these notions are undecidable in general. An in-depth analysis of the hardness of *AST* and *PAST* can be found in [KK15]. As a result, sufficient conditions (we will call them *proof rules*) for these concepts have been developed. In this chapter, we give an overview of some of these proof rules. After that, in Section 3.4, we state our first result, a generalization of a proof-rule witnessing non-*AST* which applies to Prob-solvable loops with polynomial variable updates. Following, we characterize the class of Prob-solvable loops for which the considered proof rules provide a complete decision procedure.

For the remainder of this chapter we fix a Prob-solvable loop \mathcal{L} together with its canonical filtered probability space $(\Omega, \Sigma, (\mathcal{F})_{i \in \mathbb{N}}, \mathbb{P})$. With the term *pure invariant*, we refer to an invariant in the classical deterministic sense, that means to a boolean expression holding in all states on all computation paths. Because of the way we defined the probability space corresponding to a Prob-solvable loop, this means that a pure invariant has to be true before and after every loop iteration.

3.1 Positive-Almost-Sure-Termination (PAST)

A proof rule for *PAST* first introduced in [CS13] relies on the notion of ranking-supermartingales (RSMs). The proof rule requires a SM to decrease by a fixed ϵ on average at every step. At first glimpse, it seems to resemble the concept of a ranking function that can be used to prove termination for deterministic programs. However, the theory behind the rule and the techniques necessary for proving it are quite different compared to ranking functions.

Theorem 2 (Ranking-Supermartingale-Rule (RSM-Rule) [CS13] [FFH15]). *Let $M : \mathbb{R}^m \rightarrow \mathbb{R}$ be an expression over the program variables of \mathcal{L} and $\epsilon > 0$. Moreover, let I be a pure invariant of \mathcal{L} . Assume the following conditions hold for all $i \in \mathbb{N}$:*

1. $\mathcal{G} \wedge I \implies M > 0$
(terminates when ≤ 0)
2. $\mathcal{G}_i \wedge I_i \implies \mathbb{E}(M_{i+1} - M_i \mid \mathcal{F}_i) \leq -\epsilon$
(RSM condition)

Then \mathcal{L} is PAST and $\mathbb{E}(T^{-\mathcal{G}}) < \frac{M_0}{\epsilon}$. In this case, M is called an ϵ -ranking supermartingale.

The previously discussed asymmetric 1D-random-walk, biased towards decreasing x , can be proved to be PAST with the RSM-Rule. A more complex example whose PAST can be proven using Theorem 2 is the following.

Example 4. Consider the following Prob-solvable loop:

```

y := 1
x := 10
while x > 0 do
  y := y + 1
  x := x - y2 [1/2] x + y
end

```

For the expression over program variables, we choose $M := x$. It is easy to see that $\mathcal{G} \implies M > 0$ holds, because $x > 0 \implies x > 0$. Showing $\mathcal{G}_i \wedge I_i \implies \mathbb{E}(M_{i+1} - M_i \mid \mathcal{F}_i) \leq -\epsilon$ leads to the following:

$$\begin{aligned} \mathbb{E}(M_{i+1} - M_i \mid \mathcal{F}_i) &\leq -\epsilon \\ x_i - \frac{y_i^2}{2} - \frac{y_i}{2} - x_i &\leq -\epsilon \\ \epsilon - \frac{y_i^2}{2} - \frac{y_i}{2} &\leq 0 \end{aligned}$$

Therefore, the RSM-condition holds when $\epsilon := 1$ and using $y \geq 1$ as the invariant I . As a result, we conclude by the RSM-Rule that the program in this example is PAST.

3.2 Almost-Sure-Termination (AST)

The symmetric 1D-random walk, as seen in Chapter 2, is a seemingly simple program that is AST but on average needs an infinite amount of time doing so. This means the program is not PAST. Therefore, it is out of reach for the RSM-rule. In [MMKK17], the authors introduce a new proof rule which does not require the SM to be ranking and applies to programs that are not PAST.

Theorem 3 (Supermartingale-Rule (SM-Rule) [MMKK17]). *Let $M : \mathbb{R}^m \rightarrow \mathbb{R}_{\geq 0}$ be an expression over the program variables. Let $p : \mathbb{R}_{\geq 0} \rightarrow (0, 1]$ (for probability) and $d : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{> 0}$ (for decrease) be fixed functions, both of them being monotonically decreasing (antitone) on strictly positive arguments. Moreover, let I be a pure invariant of \mathcal{L} . Assume the following conditions hold for all $i \in \mathbb{N}$:*

1. $I \wedge \mathcal{G} \implies M > 0$
(terminates when ≤ 0)
2. for all $r \in \mathbb{R}_{> 0} : I_i \wedge \mathcal{G}_i \implies \mathbb{P}(M_{i+1} \leq M_i - d(M_i) \mid \mathcal{F}_i) \geq p(M_i)$
(decreases by at least d with probability at least p)
3. $\mathcal{G}_i \wedge I_i \implies \mathbb{E}(M_{i+1} - M_i \mid \mathcal{F}_i) \leq 0$
(SM condition)

Then \mathcal{L} is AST.

The requirement of d and p being antitone rules out that the progress towards termination becomes infinitely small while still being positive. The SM-Rule can be used to certify almost-sure-termination for the symmetric 1D-random-walk in the following way.

Example 5. *Let us revisit the symmetric 1D-random walk:*

$x := x_0$

while $x > 0$ **do**

 | $x := x - 1 \lfloor \frac{1}{2} \rfloor x + 1$

end

Let $M := x$, $p := \frac{1}{2}$ and $d := 1$.

Both p and d are trivially antitone.

Moreover, the SM-Rule can be applied because:

- Condition 1 holds because $x > 0 \implies x > 0$.
- Condition 2 holds because M decreases by at least d with probability p in every iteration.

- Condition 3 holds because:

$$\begin{aligned}\mathbb{E}(M_{i+1} - M_i \mid \mathcal{F}_i) &\leq 0 \\ x_i - x_i &\leq 0\end{aligned}$$

Therefore, we can conclude using the SM-Rule that the symmetric 1D-random-walk is AST.

3.3 Non-Termination

Two sufficient conditions for certifying the negations of AST and PAST have been introduced in [CNZ17], where the proof rule for non-PAST is a slight variation of the non-AST rule. Both rules rely on the notion of a SM being *repulsing*. Intuitively, this means that some expression M on average decreases in every step but when terminating would have to be positive, so figuratively it repulses terminating states.

Theorem 4 (Repulsing-AST-Rule (R-AST-Rule) [CNZ17]). *Let $M : \mathbb{R}^m \rightarrow \mathbb{R}$ be an expression over the program variables of \mathcal{L} , $\epsilon > 0$ and $c > 0$. Moreover, let I be a pure invariant of \mathcal{L} . Assume the following conditions hold for all $i \in \mathbb{N}$:*

1. $M_0 < 0$
(negative at beginning)
2. $I \wedge \neg \mathcal{G} \implies M \geq 0$
(≥ 0 on termination)
3. $I_i \wedge \mathcal{G}_i \implies \mathbb{E}(M_{i+1} - M_i \mid \mathcal{F}_i) \leq -\epsilon$
(RSM condition)
4. $|M_{i+1} - M_i| < c$
(c -bounded differences)

Then \mathcal{L} is not AST. The expression M is called an ϵ -repulsing supermartingale with c -bounded differences.

Next is a simple program illustrating how the R-AST-Rule can be applied for certifying that a program is not AST.

Example 6. Consider the following program:

```
x := x0

while x > 0 do
  | x := x - 1 [1/2] x + 2
end
```


Choose $M := -x$ as the expression over the program variables. The R-AST-Rule can be applied because condition 1 ($-x_0 < 0$), condition 2 ($x \leq 0 \implies -x \geq 0$) and condition 4 (c -bounded difference) hold trivially. Investigating condition 3 gives us:

$$\begin{aligned}\mathbb{E}(M_{i+1} - M_i \mid \mathcal{F}_i) &\leq -\epsilon \\ -x_i - \frac{1}{2} + x_i &\leq -\epsilon \\ -\frac{1}{2} &\leq -\epsilon\end{aligned}$$

Therefore, the RSM-condition of the R-AST-Rule holds when $\epsilon := \frac{1}{2}$. As a result, we conclude that the program is not AST.

As already proved by the SM-Rule, the symmetric 1D-random-walk is AST. However, we have not yet shown that the symmetric 1D-random-walk on average requires an infinite amount of time for termination, meaning the program is not PAST. In [CNZ17] the authors also include a variation of the R-AST-Rule which can be used for certifying a program not to be PAST.

Theorem 5 (Repulsing-PAST-Rule (R-PAST-Rule) [CNZ17]). *Let $M : \mathbb{R}^m \rightarrow \mathbb{R}$ be an expression over the program variables of \mathcal{L} , $\epsilon \geq 0$ and $c > 0$. Moreover, let I be a pure invariant of \mathcal{L} . Assume the following conditions hold for all $i \in \mathbb{N}$:*

1. $M_0 < 0$
(negative at beginning)
2. $I \wedge \neg \mathcal{G} \implies M \geq 0$
(≥ 0 on termination)
3. $I_i \wedge \mathcal{G}_i \implies \mathbb{E}(M_{i+1} - M_i \mid \mathcal{F}_i) \leq -\epsilon$
(RSM condition)
4. $|M_{i+1} - M_i| < c$
(c -bounded differences)

Then \mathcal{L} is not PAST.

The only difference between the conditions for the R-AST-Rule and the R-PAST-Rule is that for the R-PAST-Rule $\epsilon \geq 0$ whereas for the R-AST-Rule $\epsilon > 0$ has to hold. With the R-PAST-Rule at hand, the symmetric 1D-random-walk can be shown not to be PAST.

Example 7. *Returning once again to the symmetric 1D-random-walk:*

```

x := x0

while x > 0 do
  | x := x - 1 [1/2] x + 1
end

```

This time choose $M := -x$, in comparison to x when certifying AST with the Supermartingale Rule. M_0 is negative. Moreover, condition 2 holds because $x \leq 0 \implies -x \geq 0$. Furthermore, M has differences bounded by 1. Regarding condition 3 of the R-PAST-Rule, we get:

$$\begin{aligned}
 \mathbb{E}(M_{i+1} - M_i \mid \mathcal{F}_i) &\leq 0 \\
 -x_i + x_i &\leq 0 \\
 0 &\leq 0
 \end{aligned}$$

Therefore, the R-PAST-Rule applies and we conclude that the symmetric 1D-random-walk is not PAST.

3.4 Polynomial Non-Termination

The R-AST-Rule, certifying non-AST for PPs, requires a given SM M to have c -bounded differences. However, in many situations, this precondition makes it unclear how to apply the proof rule to PPs containing polynomial updates without non-trivial program transformations. In [CNZ17], the proof rule was introduced for PPs with affine updates only. The following example illustrates the explained limit of the R-AST-Rule.

Example 8 (Limits of R-AST-Rule).

```

x := 10
y := 0

while x > 0 do
  | x := x + y2 [2/3] x - y2
  | y := y + 1
end

```

$M := -x$ is an ϵ -repulsing supermartingale. However, because $|M_{i+1} - M_i| = i^2$, the differences of M cannot be bounded by a global constant. Therefore, regarding non-AST, the R-AST-Rule cannot be applied to the loop with M , because it would require M to have c -bounded differences. However, the program is not AST, as we argue and prove next.

In this section we will show that a generalization of the R-AST-Rule is possible, allowing to prove non-AST of Prob-solvable loops containing polynomial updates. The basic

idea is the following: At the heart of R-AST-Rule's proof lies Azuma's inequality, a powerful concentration result for SMs. In [CNZ17] it is stated that to apply Azuma's inequality to a SM M , M has to have c -bounded differences. However, this is not the most general form of Azuma's inequality, as it only requires the differences of M to be bounded, that means $|M_{i+1} - M_i| < c_{i+1}$ (compared to $|M_{i+1} - M_i| < c$). Furthermore, $|M_{i+1} - M_i| < c_{i+1}$ is always satisfied for Prob-solvable loops: Just take all possible computation paths of length $i + 1$ and define c_{i+1} as an upper bound of $|M_{i+1} - M_i|$ over all these computation paths. As long as M is well-defined and finite this upper bound c_{i+1} exists.

Using the reasoning sketched above, we aim to generalize the R-AST-Rule by relaxing the precondition which requires c -bounded differences for M . We replace the condition of the R-AST-Rule involving c -bounded differences with a new condition. This modification will prove to be a proper generalization and allow for proving non-AST for strictly more programs. The generalization we propose in Theorem 7 is not restricted to Prob-solvable loops, but can be used on any PP. To give a preview, the new condition will require the bounds c_i to asymptotically grow only as fast as ϵ_i (that means $c_i \in O(\epsilon_i)$), where ϵ_i stems from a generalization of ϵ -repulsing supermartingales. We will start making our ideas precise by generalizing repulsing supermartingales as follows.

Definition 15 (Generalized Repulsing Supermartingale). *Let M be a supermartingale and T be a stopping time (e.g. the looping time $T^{-\mathcal{G}}$). Moreover, let $(\epsilon_i)_{i \in \mathbb{N}}$ be a sequence over \mathbb{R}^+ . M is ϵ_i -repulsing if it is ϵ_i -decreasing until T (i.e. $i < T \implies \mathbb{E}(M_{i+1} - M_i \mid \mathcal{F}_i) \leq -\epsilon_{i+1}$) as well as for all $\vartheta \in \Omega$ and $i \in \mathbb{N}$ it holds that $T(\vartheta) = i \implies M_i(\vartheta) \geq 0$.*

Our definition of ϵ_i -repulsing supermartingales is a proper generalization of [CNZ17]. When restricting the sequence $(\epsilon_i)_{i \in \mathbb{N}}$ to be constant (i.e. $\epsilon_i = \epsilon$ for all i), our ϵ_i -repulsing supermartingale yields an ϵ -repulsing supermartingale as in [CNZ17].

Intuitively, Definition 15 states that a supermartingale M is ϵ_i -decreasing if, at the i -th step, M is expected to decrease by at least ϵ_i until termination, and upon termination, M takes a non-negative value. In the following, we say that M is a *repulsing supermartingale* for a Prob-solvable loop \mathcal{L} if M is a repulsing supermartingale with respect to the looping time $T^{-\mathcal{G}}$.

Next, we state *Azuma's inequality*, a powerful concentration result for SMs [Wil91]. Intuitively it states that for a SM M , the probability of M surpassing its starting value by λ decreases exponentially in λ .

Theorem 6 (Azuma's Inequality [Wil91]). *Suppose M is a SM with bounded differences (i.e. $|M_{i+1} - M_i| < c_{i+1}$ almost surely). Then for all $n \in \mathbb{N}$ and all $\lambda > 0$ the following holds:*

$$\mathbb{P}(M_n - M_0 \geq \lambda) \leq \exp\left(\frac{-\lambda^2}{2 \sum_{k=1}^n c_k^2}\right)$$

3. PROOF RULES CERTIFYING TERMINATION PROPERTIES

With the help of Azuma's inequality, we can now effectively impose a bound on the probability that the Prob-solvable loop \mathcal{L} terminates almost surely in exactly n steps.

Lemma 1. *Let F_n be the set of all program runs of the Prob-solvable loop \mathcal{L} having looping time n , that means $F_n = \{\vartheta \in \Omega \mid T^{-\mathcal{G}}(\vartheta) = n\}$. Moreover, assume $M_0 < 0$ and for all $i \in \mathbb{N}$:*

- M is an ϵ_i -repulsing supermartingale
- M has bounded differences (bounded by c_i)
- $c_i \in O(\epsilon_i)$

Then, there is an $N_0 \in \mathbb{N}$ and $\gamma \in (0, 1)$ such that for all $n > N_0$ it holds that $\mathbb{P}(F_n) \leq \gamma^n$.

Proof. First we define a stochastic process \tilde{M} :

$$\tilde{M}_i(\vartheta) = \begin{cases} M_i(\vartheta) + \sum_{k=1}^i \epsilon_k & \text{if } i \leq T^{-\mathcal{G}}(\vartheta) \\ \tilde{M}_{i-1}(\vartheta), & \text{otherwise} \end{cases}$$

Because M is ϵ_i -decreasing, \tilde{M} is a SM. Moreover, it is easy to verify that \tilde{M} has differences bounded by $(c_i + \epsilon_i)$.

Assuming $\vartheta \in F_n$ it holds that $\tilde{M}_n(\vartheta) = M_n(\vartheta) + \sum_{k=1}^n \epsilon_k \geq \sum_{k=1}^n \epsilon_k$, because M is positive at termination and ϑ has termination time n by assumption. Subtracting $\tilde{M}_0(\vartheta)$ ($= M_0(\vartheta) = M_0$) from both sides gives us:

$$\vartheta \in F_n \implies \tilde{M}_n(\vartheta) - \tilde{M}_0(\vartheta) \geq \sum_{k=1}^n \epsilon_k - M_0$$

Therefore, we can conclude:

$$\mathbb{P}(F_n) \leq \mathbb{P}(\tilde{M}_n - \tilde{M}_0 \geq \sum_{k=1}^n \epsilon_k - M_0)$$

Now, we can use Theorem 6 to arrive at

$$\mathbb{P}(F_n) \leq \exp\left(\frac{-(\sum_{k=1}^n \epsilon_k - M_0)^2}{2 \sum_{k=1}^n (c_k + \epsilon_k)^2}\right) =: \exp\left(-\frac{1}{2}f(n)\right),$$

where

$$f(n) = \frac{(\sum_{k=1}^n \epsilon_k - M_0)^2}{\sum_{k=1}^n (c_k + \epsilon_k)^2}$$

To move forward we analyze the asymptotic behavior of the obtained bound. We start by analyzing the fraction defining $f(n)$. It holds that $(\sum_{k=1}^n \epsilon_k - M_0)^2 \in \Theta(n^2 \epsilon_n^2)$. Moreover, $\sum_{k=1}^n (c_k + \epsilon_k)^2 \in \Theta(n \epsilon_n^2)$. The last claim holds because of our assumption $c_i \in O(\epsilon_i)$. Therefore, we get $f(n) \in \Theta(n)$. Using the definition of Θ gives us:

$$\exists N_0 \in \mathbb{N}, C_1 > 0, C_2 > 0 \text{ s.t. } \forall n > N_0 : C_1 n \leq f(n) \leq C_2 n$$

Therefore, we get that for all $n > N_0$ it holds that $\mathbb{P}(F_n) \leq \exp(-Dn)$ for a constant $D > 0$. Defining γ to be e^{-D} ($\in (0, 1)$) we get our desired result:

There is an $N_0 \in \mathbb{N}$ and $\gamma \in (0, 1)$ such that for all $n > N_0$ it holds that $\mathbb{P}(F_n) \leq \gamma^n$. \square

Remark 2. Our Lemma 1 generalizes [CNZ17] - Lemma 3. In [CNZ17], ϵ_i and c_i are assumed to be constant and therefore also $c_i \in O(\epsilon_i)$.

Using Lemma 1, we now impose a bound on the probability of termination.

Lemma 2. Let F_n , M , N_0 and γ be as in Lemma 1. Then, for any $n > N_0$ the following holds:

$$\mathbb{P}(T^{-\mathcal{G}} < \infty \mid T^{-\mathcal{G}} \geq n) \leq \frac{\gamma^n}{1 - \gamma}$$

Proof.

$$\begin{aligned} \mathbb{P}(T^{-\mathcal{G}} < \infty \mid T^{-\mathcal{G}} \geq n) &= \sum_{k=n}^{\infty} \mathbb{P}(F_k) \\ &\leq \sum_{k=n}^{\infty} \gamma^k && \text{(by Lemma 1)} \\ &= \frac{\gamma^n}{1 - \gamma} \end{aligned}$$

\square

Remark 3. Note that for large enough n the bound is strictly smaller than 1.

Using basic rules of probability, we can also establish the next result.

Lemma 3. Let F_n , M , N_0 and γ be as in Lemma 1. Then, for any $n > N_0$ the following holds:

$$\mathbb{P}(T^{-\mathcal{G}} < \infty) \leq 1 + \left(\frac{\gamma^n}{1 - \gamma} - 1\right) \cdot \mathbb{P}(T^{-\mathcal{G}} \geq n)$$

Proof.

$$\begin{aligned}
 \mathbb{P}(T^{-\mathcal{G}} < \infty) &= \mathbb{P}(T^{-\mathcal{G}} < \infty \mid T^{-\mathcal{G}} \geq n) \cdot \mathbb{P}(T^{-\mathcal{G}} \geq n) \\
 &\quad + \mathbb{P}(T^{-\mathcal{G}} < \infty \mid T^{-\mathcal{G}} < n) \cdot \mathbb{P}(T^{-\mathcal{G}} < n) \\
 &= \mathbb{P}(T^{-\mathcal{G}} < \infty \mid T^{-\mathcal{G}} \geq n) \cdot \mathbb{P}(T^{-\mathcal{G}} \geq n) + \mathbb{P}(T^{-\mathcal{G}} < n) \\
 &\leq \frac{\gamma^n}{1-\gamma} \cdot \mathbb{P}(T^{-\mathcal{G}} \geq n) + \mathbb{P}(T^{-\mathcal{G}} < n) && \text{(by Lemma 2)} \\
 &= \frac{\gamma^n}{1-\gamma} \cdot \mathbb{P}(T^{-\mathcal{G}} \geq n) + 1 - \mathbb{P}(T^{-\mathcal{G}} \geq n) \\
 &= 1 + \left(\frac{\gamma^n}{1-\gamma} - 1\right) \cdot \mathbb{P}(T^{-\mathcal{G}} \geq n)
 \end{aligned}$$

□

Note that for large enough n , the bound in Lemma 3 is strictly smaller than 1 if $\mathbb{P}(T^{-\mathcal{G}} \geq n)$ is non-zero. To establish this we will prove that under some assumptions, such as assuming the existence of a repulsing supermartingale, $\mathbb{P}(T^{-\mathcal{G}} \geq n) > 0$ for all $n \in \mathbb{N}$.

Lemma 4. *Assume M is a ϵ_i -repulsing supermartingale and $M_0 < 0$. Then, for all $n \in \mathbb{N}$ it holds that $\mathbb{P}(T^{-\mathcal{G}} > n) > 0$.*

Proof. Let $n \in \mathbb{N}$ be arbitrary. We define a stopping time \tilde{T} by

$$\tilde{T}(\vartheta) = \inf\{i \in \mathbb{N} \mid \vartheta[i] \vDash \neg\mathcal{G} \text{ or } i \geq n\}.$$

This means, for a given run of the program ϑ , $\tilde{T}(\vartheta)$ is the first point in time when the loop guard is falsified or n is reached or surpassed. \tilde{T} is bounded by n , which means the Optional Stopping Theorem (Theorem 1) applies, giving us:

$$\mathbb{E}(M_{\tilde{T}}) \leq \mathbb{E}(M_0) < 0$$

Towards a contradiction, assume $\mathbb{P}(T^{-\mathcal{G}} > n) = 0$. Then it follows that at time \tilde{T} the program is almost surely in a state falsifying the loop guard \mathcal{G} . By definition of repulsing supermartingales this entails that $M_{\tilde{T}} \geq 0$ almost surely. However, this contradicts $\mathbb{E}(M_{\tilde{T}}) < 0$, and we conclude $\mathbb{P}(T^{-\mathcal{G}} > n) > 0$. □

Now we have all ingredients to prove our main theorem yielding a new proof rule for non-AST.

Theorem 7 (Generalized-Repulsing-AST-Rule (GR-AST-Rule)). *A Prob-solvable loop \mathcal{L} is not AST, if $M_0 < 0$ and the following conditions are true for all $i \in \mathbb{N}$:*

- M is an ϵ_i -repulsing supermartingale with respect to $T^{-\mathcal{G}}$
- M has bounded differences (bounded by c_i)
- $c_i \in O(\epsilon_i)$

Proof. By Lemma 3, we know that

$$\mathbb{P}(T^{-\mathcal{G}} < \infty) \leq 1 + \left(\frac{\gamma^n}{1-\gamma} - 1\right) \cdot \mathbb{P}(T^{-\mathcal{G}} \geq n)$$

for some $\gamma \in (0, 1)$ and all n bigger than some N_0 . The bound is strictly smaller than 1 for large enough n if $\mathbb{P}(T^{-\mathcal{G}} \geq n) > 0$.

Lemma 4 tells us that $\mathbb{P}(T^{-\mathcal{G}} \geq n) > 0$ is true for any n . Therefore, we can conclude that $\mathbb{P}(T^{-\mathcal{G}} < \infty) < 1$. \square

As already mentioned, Theorem 7 generalizes [CNZ17]. The result of [CNZ17] can be obtained from our result by restricting ϵ_i and c_i to be constant. To show that it is a proper generalization, we will revisit Example 8 for which the result in [CNZ17] is not directly applicable. However, our GR-AST-Rule from Theorem 7 applies and certifies that Example 8 is not AST.

Example 9.

$x := 10$

$y := 0$

while $x > 0$ **do**

$x := x + y^2 \quad \left[\frac{2}{3}\right] \quad x - y^2$

$y := y + 1$

end

Choosing $M := -x$ we have:

- $\mathbb{E}(M_{i+1} - M_i \mid \mathcal{F}_i) = -\frac{i^2}{3}$
- On termination M is non-negative
- $|M_{i+1} - M_i| = i^2$
- $M_0 = -10 < 0$

Therefore, all preconditions of Theorem 7 are satisfied: M is a repulsing supermartingale with $\epsilon_i = \frac{i^2}{3}$. The differences of M are bounded by $c_i = i^2$. Thus, also $c_i \in O(\epsilon_i)$ holds. Therefore, we conclude that the program is not AST.

3.5 Relaxations of Proof Rules

When considering PPs containing polynomial updates, an important relaxation of the termination proof rules is to allow for the conditions of the proof rules to only hold eventually. A property $P(i)$ parameterized by a natural number $i \in \mathbb{N}$ holds *eventually* means, there is an $i_0 \in \mathbb{N}$ such that $P(i)$ holds for all $i \geq i_0$. Informally, if a PP, after a fixed number of steps, almost surely reaches a state from which the program is PAST (AST), then the whole program is PAST (AST). The following two examples illustrate this fact for PAST and AST.

Example 10 (Limits RSM-Rule).

```

 $x := x_0$ 
 $n := 0$ 

while  $x > 0$  do
  |  $n := n + 1$ 
  |  $x := x + 4n \ [\frac{1}{2}] \ x - n^2$ 
end

```

For the loop and x , we have the martingale expression $\mathbb{E}(x_{i+1} - x_i \mid \mathcal{F}_i) = -\frac{i^2}{2} + i + \frac{3}{2}$. Therefore, x cannot be a RSM, because for $i \in \{0, 1, 2, 3\}$ the expression is non-negative. However, the program either terminates within the first three iterations or after three iterations is in a state such that the RSM-Rule is applicable. Therefore, the whole program is PAST.

Example 11 (Limits SM-Rule).

```

 $x := x_0$ 
 $n := 0$ 

while  $x > 0$  do
  |  $n := n + 1$ 
  |  $x := x + (n - 5) \ [\frac{1}{2}] \ x - (n - 5)$ 
end

```

The martingale expression for the loop and x is $\mathbb{E}(x_{i+1} - x_i \mid \mathcal{F}_i) = 0$, meaning x is a martingale. However, defining the decrease function d for the SM-Rule cannot be done, because for example in the 5th loop iteration there is no progress at all. The variable x just gets updated with its previous value.

Yet, after the 5th iteration, x always decreases by at least 1 with probability $\frac{1}{2}$. Therefore, all conditions for the SM-Rule are satisfied and the rule certifies the loop to be AST from that point onward. Moreover, the whole loop either terminates in the first 5 steps or reaches a state from which it terminates almost surely. Consequently, the whole loop is AST.

We capture the relaxations of the RSM-Rule and the SM-Rule illustrated in Example 10 and Example 11 with the following theorem.

Theorem 8 (Relaxation Termination Rules). *For the RSM-Rule to certify PAST (Theorem 2), as well as for the SM-Rule to certify AST (Theorem 3), it is sufficient for the conditions to hold eventually (instead of all $i \in \mathbb{N}$).*

Proof sketch. Given the Prob-solvable loop $\mathcal{L} = \mathcal{I} \text{ while } \mathcal{G} \text{ do } \mathcal{U} \text{ end}$, satisfying the conditions for the RSM-Rule (the SM-Rule) after some $i_0 \in \mathbb{N}$. Construct the following probabilistic program \mathcal{P} , where i is a new variable not appearing in \mathcal{L} :

```

 $\mathcal{I}; i := 0$ 
while  $i < i_0$  do
  |  $\mathcal{U}; i := i + 1$ 
end
while  $\mathcal{G}$  do
  |  $\mathcal{U}$ 
end

```

To begin with, we observe that by construction of \mathcal{P} , if \mathcal{P} is PAST (AST) then \mathcal{L} is PAST (AST): Assume \mathcal{P} is PAST (AST). Then by construction of \mathcal{P} , \mathcal{L} 's looping time is either bounded by i_0 or it is PAST (AST). In both cases \mathcal{L} is PAST (AST).

Now, \mathcal{P} is PAST (AST) if and only if its second while-loop is. For the second while-loop PAST (AST) can be certified using the RSM-Rule (SM-Rule) and additionally using $i \geq i_0$. \square

In the remainder of this work, when referring to the RSM-Rule or the SM-Rule we refer to the respective proof rule together with its relaxation stated in Theorem 8.

We establish a similar relaxation for non-termination proof rules. However, in comparison to the termination proof rules, it is not enough for a non-termination proof rule to certify non-AST from some point onward, because the program may never reach this point and always terminate earlier. Therefore, it has to be additionally assumed that the program has a positive probability of reaching the point after which a proof rule witnesses non-AST. The following example illustrates this idea.

Example 12 (Limits GR-AST-Rule).

```

x := 1
n := 0
while x > 0 do
  | n := n + 1
  | x := x + n2 - 1  $\lfloor \frac{2}{3} \rfloor$  x - n2 + 1
end
    
```

Calculating the martingale expression for the loop and $M := -x$ leaves us with $\mathbb{E}(M_{i+1} - M_i \mid \mathcal{F}_i) = -\frac{i^2}{3} - \frac{2i}{3}$. For $i = 0$, the expression is 0 and so $-x$ cannot be an ϵ_i -repulsing supermartingale such that $e_i > 0$ for all $i \in \mathbb{N}$. However, after the first iteration, $-x$ satisfies all requirements of an ϵ_i -repulsing supermartingale and also all other conditions of the GR-AST-Rule hold. Moreover, the loop always reaches the second iteration because in the first iteration x does not change at all. From this follows that the loop is not AST.

We capture the concept illustrated in Example 12 with the following theorem.

Theorem 9 (Relaxation Non-Termination Rules). *For the GR-AST-Rule to certify non-AST (Theorem 7), if $\mathbb{P}(T^{-\mathcal{G}} > i_0) > 0$ for some $i_0 \geq 0$, it suffices that $M_{i_0} < 0$ holds instead of $M_0 < 0$. For the remaining conditions it suffices for them to hold for all $i \geq i_0$ (instead of for all $i \in \mathbb{N}$).*

Proof sketch. Given the Prob-solvable loop $\mathcal{L} = \mathcal{I}$ while \mathcal{G} do \mathcal{U} end, assume all conditions of the GR-AST-Rule to be satisfied for all $i \geq i_0$ for some fixed i_0 . Moreover, assume $\mathbb{P}(T^{-\mathcal{G}} > i_0) > 0$.

We repeat the same construction as in the proof of Theorem 8 and get the following probabilistic program \mathcal{P} :

```

 $\mathcal{I}; i := 0$ 
while  $i < i_0$  do
  |  $\mathcal{U}; i := i + 1$ 
end
while  $\mathcal{G}$  do
  |  $\mathcal{U}$ 
end
    
```

The GR-AST-Rule certifies the second while-loop of \mathcal{P} not being AST, using that \mathcal{L} satisfies all conditions of the proof rule for $i \geq i_0$ and the fact that for the second while loop $i \geq i_0$. From this and by the definition of \mathcal{P} it follows that if there is a $Cyl(\pi) \in \mathcal{F}_{i_0}^{\mathcal{L}}$ such that $\mathbb{P}^{\mathcal{L}}(Cyl(\pi)) > 0$ and $T^{-\mathcal{G}}(\vartheta) > i_0$ for all $\vartheta \in Cyl(\pi)$, then

$\mathbb{P}(T^{-\mathcal{G}} = \infty) > 0$. By the assumption that $\mathbb{P}(T^{-\mathcal{G}} > i_0) > 0$ such a $Cyl(\pi)$ exists, and we get that $\mathbb{P}^{\mathcal{L}}(T^{-\mathcal{G}} = \infty) > 0$, meaning \mathcal{L} is non-AST. \square

As with the proof rules for AST and PAST, in the remainder of this work, when referring to the GR-AST-Rule, we refer to the proof rule together with its relaxation stated in Theorem 9.

3.6 Completeness

Having established proof rules for AST as well as for non-AST, a natural question arising is, for what class of PPs these proof rules constitute a complete decision procedure. Is there a class of Prob-solvable loops whose AST can be decided effectively? In this section, we will positively answer this question.

In [GGH19] the authors mention the RSM-Rule, the SM-Rule and the R-AST-Rule constituting a complete decision procedure for so-called *constant probability programs*. The authors provide a transformation of constant probability programs to random walk programs preserving the termination behavior and supply a decision procedure for the termination of random walk programs.

Informally, a random walk program contains a single variable x , consists of an initialization part and a while-loop with the loop guard $x > 0$. The loop body is a single probabilistic update of the form $x := x + c_1 [p_1] \dots [p_l] x + c_{l+1}$, where c_j and p_j are constants. Random walk programs fall into the class of Prob-solvable loops. Using our GR-AST-Rule instead of the R-AST-Rule, we provide a decision procedure deciding AST for a larger class of programs than in [GGH19].

Instead of allowing constants to be added to a variable x , we allow the product of polynomials in the loop counter with an exponential function. The product $poly(i) \cdot b^i$ for $b \geq 1$ cannot converge to 0 without being identical to 0, because this property is true for polynomials and b^i is always positive. Restricting the updates to the variable x to this sort of functions is particularly useful because figuratively it rules out the behavior that the progress towards termination is positive but becomes smaller and smaller. This leads us to the following class of Prob-solvable loops.

Definition 16 (Poly-Exponential Random-Walk). *A Prob-solvable loop \mathcal{L} is a poly-exponential random-walk (PE-RW) if it has the form*

```

 $x := x_0$ 
 $i := 0$ 
 $e := 1$ 
while  $x > 0$  do
   $i := i + 1$ 
   $e := b \cdot e$ 
   $x := x + q_1(i) \cdot e [p_1] \dots [p_l] x + q_{l+1}(i) \cdot e$ 
end

```

where $l \geq 0$, $x_0 \in \mathbb{R}$, $b \geq 1$ and every $q_j \in \mathbb{R}[i]$ is a polynomial. The polynomials q_j are assumed to be pairwise different. The loop \mathcal{L} is called trivial if $l = 0$ and $q_1 \equiv 0$, which means the update for x has the form $x := x + 0 [1]$.

The functions $\mathbb{U}_{\mathcal{L}} := \{q_j(i) \cdot b^i \mid 1 \leq j \leq l+1\}$ are called \mathcal{L} 's updates. With $\mathcal{E}_{\mathcal{L}}(i)$ we denote the function

$$\mathcal{E}_{\mathcal{L}}(i) := \left[\sum_{j=1}^{l+1} p_j \cdot q_j(i+1) \right] \cdot b^{i+1}. \quad (3.1)$$

For a PE-RW \mathcal{L} , the function $\mathcal{E}_{\mathcal{L}}(i)$ is the martingale expression for the variable x :

$$\begin{aligned} \mathbb{E}(x_{i+1} - x_i \mid \mathcal{F}_i) &= \left[\sum_{j=1}^{l+1} p_j \cdot (x_i + q_j(i+1) \cdot e_{i+1}) \right] - x_i \\ &= \sum_{j=1}^{l+1} p_j \cdot q_j(i+1) \cdot e_{i+1} \\ &= \sum_{j=1}^{l+1} p_j \cdot q_j(i+1) \cdot b^{i+1} \\ &= \left[\sum_{j=1}^{l+1} p_j \cdot q_j(i+1) \right] \cdot b^{i+1} \\ &= \mathcal{E}_{\mathcal{L}}(i) \end{aligned}$$

Moreover, the martingale expression $\mathcal{E}_{\mathcal{L}}(i)$ is the product of the polynomial $poly(i) = \sum_{j=1}^{l+1} p_j \cdot q_j(i+1)$ and b^{i+1} . Therefore, the zeros of $\mathcal{E}_{\mathcal{L}}(i)$ correspond to the zeros of $poly(i)$ and hence are computable. In addition, the limit $\lim_{i \rightarrow \infty} \mathcal{E}_{\mathcal{L}}(i)$ is computable and equals:

- 0, if $\mathcal{E}_{\mathcal{L}} \equiv 0$;

- $+\infty$, if $\lim_{i \rightarrow \infty} \text{poly}(i) = +\infty$ or $\text{poly}(i) \in \mathbb{R}^+$ and $b > 1$;
- $-\infty$, if $\lim_{i \rightarrow \infty} \text{poly}(i) = -\infty$ or $\text{poly}(i) \in \mathbb{R}^-$ and $b > 1$;
- $c \in \mathbb{R}$, if $\text{poly} \equiv c$ and $b = 1$.

Our approach for deciding AST for PE-RWs is to first check whether a PE-RW \mathcal{L} immediately terminates ($x_0 \leq 0$), in which case \mathcal{L} is AST, or whether \mathcal{L} is trivial, in which case \mathcal{L} is not AST. If none of those cases apply, we consider the limit of $\mathcal{E}_{\mathcal{L}}(i)$. If the limit is positive, x eventually is a RSM, and we can certify AST with the RSM-Rule. If the limit is zero, x is a SM, and AST is certified by the SM-Rule. If the limit is negative, $-x$ eventually is an ϵ_i -repulsing supermartingale after some iteration i_0 . In this case, the loop \mathcal{L} is only non-AST by the GR-AST-Rule if the i_0 -th iteration can be reached with positive probability and the condition $c_i \in O(\epsilon_i)$ of the GR-AST-Rule is satisfied. However, the condition $c_i \in O(\epsilon_i)$ is not necessarily satisfied for PE-RWs as the following example illustrates.

Example 13. Consider the following PE-RW:

```

x := x0
i := 0
while x > 0 do
  | i := i + 1
  | x := x + i3 + 2i2 [½] x - i3 - i2
end

```

Computing the expression $\mathcal{E}_{\mathcal{L}}(i)$ leads to: $\mathcal{E}_{\mathcal{L}}(i) = \frac{i^2}{2} + i + \frac{1}{2}$. Therefore, $-x$ is an ϵ_i -repulsing supermartingale with $\epsilon_i \in \Theta(i^2)$. However, the bounds c_i on $|x_{i+1} - x_i|$ are of order $\Theta(i^3)$. Consequently, the condition $c_i \in O(\epsilon_i)$ of the GR-AST-Rule does not hold and the proof rule cannot be applied.

Therefore, we have to further restrict our class for which we give a complete decision procedure for AST to ensure that the condition $c_i \in O(\epsilon_i)$ of the GR-AST-Rule holds when necessary.

Definition 17 (Admissible PE-RW). A PE-RW \mathcal{L} is called admissible if $\lim_{i \rightarrow \infty} \mathcal{E}_{\mathcal{L}}(i) > 0$ implies $\Theta(\mathcal{E}_{\mathcal{L}}(i)) = \max\{\Theta(f(i)) \mid f \in \mathbb{U}_{\mathcal{L}}\}$.

The maximum is with respect to the order $f(i) \leq g(i) \iff f(i) \in O(g(i))$.

The notion of a PE-RW being admissible ensures that if the limit of $\mathcal{E}_{\mathcal{L}}(i)$ is positive and therefore $-x$ eventually is an ϵ_i -repulsing supermartingale, the dominating term among all $f \in \mathbb{U}_{\mathcal{L}}$ does not cancel out when computing $\mathcal{E}_{\mathcal{L}}(i)$. This ensures, the condition $c_i \in O(\epsilon_i)$ of the GR-AST-Rule to always be satisfied when it is needed. Note, that verifying a PP

to be a PE-RW can be done by purely syntactic means. Moreover, checking a PE-RW to be admissible amounts to the simple computation of $\lim_{i \rightarrow \infty} \mathcal{E}_{\mathcal{L}}(i)$ and comparing the dominant term of $\mathcal{E}_{\mathcal{L}}(i)$ with the dominant terms of all $f \in \mathbb{U}_{\mathcal{L}}$. We note that all random-walk programs as defined in [GGH19] fall into the class of admissible PE-RWs. Yet, not all admissible PE-RWs are captured by [GGH19]. For instance, Example 14 and Example 15 show limitations of [GGH19] whereas our work decides their AST.

Our approach deciding AST for admissible PE-RWs is summarized in Algorithm 3.1.

Algorithm 3.1: Deciding AST for admissible PE-RWs

Input: An admissible PE-RW \mathcal{L}
Output: *true* if \mathcal{L} is AST; *false* otherwise

```

1 if  $x_0 \leq 0$  then
2   | return true
3 end
4 if  $\mathcal{L}$  is trivial then
5   | return false
6 end
7 limit :=  $\lim_{i \rightarrow \infty} \mathcal{E}_{\mathcal{L}}(i)$ 
8 if limit  $\leq 0$  then
9   | return true
10 end
11 max0 :=  $\lceil \max(\{r \in \mathbb{R} \mid \mathcal{E}_{\mathcal{L}}(r) = 0\} \cup \{0\}) \rceil$ 
12 M :=  $x_0 + \sum_{i=1}^{\text{max0}} \max\{f(i) \mid f \in \mathbb{U}_{\mathcal{L}}\}$ 
13 if M  $\leq 0$  then
14   | return true
15 else
16   | return false
17 end

```

The termination of Algorithm 3.1 is trivial: There are no loops present in the algorithm. The limit on line 7 as well as the calculations of the zeros of $\mathcal{E}_{\mathcal{L}}$ on line 11 are computable, as previously argued. Moreover, the sum in line 12 is finite, and also the maximum is over finitely many values. The correctness of the algorithm is proved next.

Theorem 10 (Correctness of Algorithm 3.1). *If Algorithm 3.1 terminates on input \mathcal{L} with return value v , then v is true if and only if \mathcal{L} is AST.*

Proof. We make a case distinction at which line Algorithm 3.1 terminates.

Algorithm 3.1 terminates at line 2:

By the definition of the loop measure \mathbb{P} and the looping time $T^{-\mathcal{G}}$, we have $\mathbb{P}(T^{-\mathcal{G}} = 0) = 1$ and therefore \mathcal{L} is PAST, hence also AST. \square

Algorithm 3.1 terminates at line 5:

We have $x_0 > 0$. Moreover, by the defining criteria of *trivial*, the update of x in the loop-body $\mathcal{U}_{\mathcal{L}}$ is $x := x + 0$ [1]. Then by the definition of the loop measure \mathbb{P} and the looping time $T^{-\mathcal{G}}$, we deduce $\mathbb{P}(T^{-\mathcal{G}} = \infty) = 1$, meaning \mathcal{L} is not AST.

Algorithm 3.1 terminates at line 9:

We have $x_0 > 0$ and \mathcal{L} is not trivial. Consider the following two cases:

1. $\lim_{i \rightarrow \infty} \mathcal{E}_{\mathcal{L}}(i) < 0$: Then x is eventually a RSM and we conclude by the RSM-Rule that \mathcal{L} is PAST and therefore also AST.
2. $\lim_{i \rightarrow \infty} \mathcal{E}_{\mathcal{L}}(i) = 0$: In this case, as previously established, $\mathcal{E}_{\mathcal{L}} \equiv 0$ and therefore x is a martingale. Because, \mathcal{L} is not trivial, there is at least one update function $q_j(i) \cdot b^i \in \mathbb{U}_{\mathcal{L}}$ such that $\lim_{i \rightarrow \infty} q_j(i) \cdot b^i < 0$. That is because, if all $f \in \mathbb{U}_{\mathcal{L}}$ would have a limit ≥ 0 , \mathcal{L} would either have to be trivial or $\mathcal{E}_{\mathcal{L}}$'s limit would necessarily be > 0 which is not possible.

Therefore, eventually x decreases by at least some constant d with probability at least p_j , which is also constant. Consequently, we conclude by the SM-Rule that \mathcal{L} is AST.

Algorithm 3.1 terminates at line 14:

We have $x_0 > 0$, $\lim_{i \rightarrow \infty} \mathcal{E}_{\mathcal{L}}(i) > 0$ and $M \leq 0$ for $M = x_0 + \sum_{i=1}^{max0} \max\{f(i) \mid f \in \mathbb{U}_{\mathcal{L}}\}$ and $max0 \in \mathbb{N}$. By the definition of the loop measure \mathbb{P} , for all cylinder sets $Cyl(\pi) \in \mathcal{F}_{max0}$ such that $\mathbb{P}(Cyl(\pi)) > 0$, it holds that $\pi_{max0}[x] \leq M \leq 0$. Therefore, $\mathbb{P}(T^{-\mathcal{G}} \leq max0) = 1$, meaning \mathcal{L} is PAST and hence also AST.

Algorithm 3.1 terminates at line 16:

We have $x_0 > 0$, $\lim_{i \rightarrow \infty} \mathcal{E}_{\mathcal{L}}(i) > 0$ and $M > 0$ for $M = x_0 + \sum_{i=1}^{max0} \max\{f(i) \mid f \in \mathbb{U}_{\mathcal{L}}\}$ and $max0 = \lceil \max(\{r \in \mathbb{R} \mid \mathcal{E}_{\mathcal{L}}(r) = 0\} \cup \{0\}) \rceil$.

Because $\lim_{i \rightarrow \infty} \mathcal{E}_{\mathcal{L}}(i) > 0$, the expression $-x$ is an ϵ_i -repulsing supermartingale after $max0$. The fact that $M > 0$ witnesses that there is a $Cyl(\pi) \in \mathcal{F}_{max0}$ such that $\mathbb{P}(Cyl(\pi)) > 0$ and $T^{-\mathcal{G}}(\vartheta) > max0$ for all $\vartheta \in Cyl(\pi)$. Therefore, $\mathbb{P}(T^{-\mathcal{G}} > max0) > 0$. Moreover, because \mathcal{L} is admissible, the condition $c_i \in O(\epsilon_i)$ of the GR-AST-Rule is satisfied. Therefore, we conclude by the GR-AST-Rule that \mathcal{L} is not AST. \square

The class of admissible PE-RWs contains various programs of interest. Additionally to the random walk programs, as defined in [GGH19], the class also contains random walk programs where the step size is defined by a monomial or by an exponential.

Example 14.

$x := x_0$ $i := 0$ while $x > 0$ do <table style="border-left: 1px solid black; border-right: 1px solid black; padding: 0 10px;"> <tr> <td style="padding: 0 5px;">$i := i + 1$</td> </tr> <tr> <td style="padding: 0 5px;">$x := x + c_1 \cdot i^k$ $[p]$ $x + c_2 \cdot i^{k'}$</td> </tr> </table> end	$i := i + 1$	$x := x + c_1 \cdot i^k$ $[p]$ $x + c_2 \cdot i^{k'}$	$x := x_0$ $i := 0$ $e := 1$ while $x > 0$ do <table style="border-left: 1px solid black; border-right: 1px solid black; padding: 0 10px;"> <tr> <td style="padding: 0 5px;">$i := i + 1$</td> </tr> <tr> <td style="padding: 0 5px;">$e := b \cdot e$</td> </tr> <tr> <td style="padding: 0 5px;">$x := x + c_1 \cdot e$ $[p]$ $x + c_2 \cdot e$</td> </tr> </table> end	$i := i + 1$	$e := b \cdot e$	$x := x + c_1 \cdot e$ $[p]$ $x + c_2 \cdot e$
$i := i + 1$						
$x := x + c_1 \cdot i^k$ $[p]$ $x + c_2 \cdot i^{k'}$						
$i := i + 1$						
$e := b \cdot e$						
$x := x + c_1 \cdot e$ $[p]$ $x + c_2 \cdot e$						

Both programs fall into the class of admissible PE-RWs, where $c_1, c_2 \in \mathbb{R}$, $k, k' \in \mathbb{N}$ and $b \geq 1$. Therefore, their termination behavior can be decided by Algorithm 3.1.

Example 15 (Gambling Strategy). *The following program models the gambling strategy of a player playing a fair game with two possible outcomes. The player has an infinite amount of wealth and the goal of winning 1 \$ in total. She has the strategy to always double her previous bet as long as she loses and stops as soon as she wins. The program is an admissible PE-RW and can be verified to be AST by Algorithm 3.1.*

$$goal := 1$$

$$bet := \frac{1}{2}$$
while $goal > 0$ **do**

$bet := 2 \cdot bet$
$goal := goal - bet$ $[\frac{1}{2}]$ $goal + bet$

end

Changing the probability of winning from $\frac{1}{2}$ to $\frac{1}{2} - \epsilon$ for some $\epsilon > 0$, changes the termination behavior of the program to non-AST, which can also be computed by Algorithm 3.1.

Remark 4. In [GGH19] the authors provide a transformation from constant probability programs to random walk programs preserving the termination behavior of the transformed programs. A constant probability program contains multiple variables and its loop guard is a linear inequality. In comparison, a random walk program contains only a single variable x and has loop guard $x > 0$.

A similar transformation could be developed for our setting of PE-RWs. The class of PE-RWs could be extended to allow for more than a single variable x and linear inequalities as loop guards. A program from the extended class could then be transformed to a PE-RW with a single variable x with the same termination behavior, using ideas motivated by [GGH19]. We leave this for future work.

Automating Termination Analysis of Probabilistic Programs

Arguably, the two major challenges when automating the proof rules introduced in Chapter 3 are (i) settling on an expression M over the program variables and (ii) proving an inequality involving $\mathbb{E}(M_{i+1} - M_i \mid \mathcal{F}_i)$. Considering the GR-AST-Rule, the martingale expression needs additionally to be bounded by a sequence ($\mathbb{E}(M_{i+1} - M_i \mid \mathcal{F}_i) \leq -\epsilon_{i+1}$), in comparison to the RSM-Rule and the SM-Rule for which the martingale expression only needs to be bounded by a constant ($\mathbb{E}(M_{i+1} - M_i \mid \mathcal{F}_i) \leq -\epsilon$ and $\mathbb{E}(M_{i+1} - M_i \mid \mathcal{F}_i) \leq 0$). In this chapter, we address these two challenges for Prob-solvable loops.

For the loop guard $\mathcal{G}_{\mathcal{L}} = P > Q$ of a Prob-solvable loop \mathcal{L} , we denote with $G_{\mathcal{L}}$ the polynomial $P - Q$. For enhanced legibility, if \mathcal{L} is irrelevant or clear from context, we omit the subscript \mathcal{L} . It holds that $G > 0$ is equivalent to \mathcal{G} . For a Prob-solvable loop, the polynomial G is a natural candidate for the expression M in termination proof rules (RSM-Rule, SM-Rule) and $-G$ for the expression M in the non-termination proof rules (GR-AST-Rule, R-PAST-Rule):

In this thesis, we address the aforementioned challenge (i) arising from automating termination analysis of Prob-solvable loops, by choosing the expression M to be G for the termination proof rules and $-G$ for the non-termination proof rules. That is, we set $M := G$ for the RSM-Rule as well as the SM-Rule and $M := -G$ for the GR-AST-Rule as well as the R-PAST-Rule. The property $\mathcal{G} \implies G > 0$, that is one precondition of the RSM-Rule as well as the SM-Rule, holds trivially. Moreover, for the GR-AST-Rule and R-PAST-Rule the preconditions $\neg\mathcal{G} \implies -G \geq 0$ and $-G_0 < 0$ are always satisfied, assuming the loop's initial state does not violate the loop guard.

The remaining preconditions of the proof rules are the following:

- RSM-Rule:

$$* \mathcal{G}_i \implies \mathbb{E}(G_{i+1} - G_i \mid \mathcal{F}_i) \leq -\epsilon \text{ for some } \epsilon > 0$$

- SM-Rule

$$* \mathcal{G}_i \implies \mathbb{E}(G_{i+1} - G_i \mid \mathcal{F}_i) \leq 0$$

$$* \mathcal{G}_i \implies \mathbb{P}(G_{i+1} \leq G_i - d \mid \mathcal{F}_i) \geq p \text{ for some } p \in (0, 1] \text{ and } d \in \mathbb{R}^+. \text{ For the purpose of efficient automation, in this chapter we restrict the functions } d(r) \text{ and } p(r) \text{ from the SM-Rule to be constant.}$$

- GR-AST-Rule

$$* \mathcal{G}_i \implies \mathbb{E}(-G_{i+1} + G_i \mid \mathcal{F}_i) \leq -\epsilon_{i+1} \text{ for a sequence } (\epsilon_i)_{i \in \mathbb{N}} \text{ over } \mathbb{R}^+$$

$$* c_i \in O(\epsilon_i) \text{ where } (c_i)_{i \in \mathbb{N}} \text{ are a bounds on the absolute differences of } G \text{ } (|G_{i+1} - G_i| \leq c_i)$$

All non-trivial preconditions express bounds over the stochastic process G_i . Namely, either bounds for $\mathbb{E}(G_{i+1} - G_i \mid \mathcal{F}_i)$ or for $G_{i+1} - G_i$ have to be established. The martingale expression $\mathbb{E}(G_{i+1} - G_i \mid \mathcal{F}_i)$ is an expression over program variables, whereas $G_{i+1} - G_i$ cannot be interpreted as a single expression but through a distribution of expressions.

Definition 18 (One-step Distribution). *For a Prob-solvable loop \mathcal{L} and an expression H over the program variables of \mathcal{L} , let $\mathcal{U}_{\mathcal{L}}^H$ denote the distribution $E \mapsto \mathbb{P}(H_{i+1} = E \mid \mathcal{F}_i)$. $\mathcal{U}_{\mathcal{L}}^H$ is called the one-step distribution of H .*

We denote with $\text{supp}(\mathcal{U}_{\mathcal{L}}^H)$ the support of the distribution $\mathcal{U}_{\mathcal{L}}^H$, that means $\text{supp}(\mathcal{U}_{\mathcal{L}}^H) := \{B \mid \mathcal{U}_{\mathcal{L}}^H(B) > 0\}$. We refer to expressions $B \in \text{supp}(\mathcal{U}_{\mathcal{L}}^H)$ by branches of H .

The notation $\mathcal{U}_{\mathcal{L}}^H$ is chosen to suggest that the loop body $\mathcal{U}_{\mathcal{L}}$ is "applied" to the expression H , leading to a distribution of expressions. Intuitively, the support $\text{supp}(\mathcal{U}_{\mathcal{L}}^H)$ of an expression H contains all possible updates of H when executing a single iteration. Following is an example stating $\mathcal{U}_{\mathcal{L}}^H$ for a loop \mathcal{L} and an expression H .

Example 16 (One-step Distribution). *Consider the following Prob-solvable loop:*

$x := 1$

$y := 1$

while $x > 0$ **do**

$y := y + 1$ $\left[\frac{1}{2} \right]$ $y + 2$
 $x := x + y$ $\left[\frac{1}{3} \right]$ $x - y$

end

For the expression $H := x^2$, the one-step distribution $\mathcal{U}_{\mathcal{L}}^H$ is as follows:

<i>Expression E</i>	<i>Probability $\mathcal{U}_{\mathcal{L}}^H(E)$</i>
$x_i^2 + 2x_iy_i + 2x_i + y_i^2 + 2y_i + 1$	$\frac{1}{6}$
$x_i^2 + 2x_iy_i + 4x_i + y_i^2 + 4y_i + 4$	$\frac{1}{6}$
$x_i^2 - 2x_iy_i - 2x_i + y_i^2 + 2y_i + 1$	$\frac{1}{3}$
$x_i^2 - 2x_iy_i - 4x_i + y_i^2 + 4y_i + 4$	$\frac{1}{3}$
<i>Any other E</i>	0

The first entry in the table can be derived as follows:

$$\begin{aligned}
 x_{i+1}^2 &= (x_i + y_{i+1})^2 = x_i^2 + 2x_iy_{i+1} + y_{i+1}^2 && \text{with probability } \frac{1}{3} \\
 &= x_i^2 + 2x_i(y_i + 1) + (y_i + 1)^2 && \text{with probability } \frac{1}{2} \cdot \frac{1}{3} \\
 &= x_i^2 + 2x_iy_i + 2x_i + y_i^2 + 2y_i + 1 && \text{with probability } \frac{1}{6}
 \end{aligned}$$

Ultimately, to automate the termination analysis with the proof rules from Chapter 3 for Prob-solvable loops, we need to be able to compute bounds for the expression $\mathbb{E}(G_{i+1} - G_i \mid \mathcal{F}_i)$ as well as for branches of G . Moreover, because of the relaxations of the proof rules established in Section 3.5, only asymptotic bounds are needed, that means bounds which hold eventually.

In Section 4.2, we propose a procedure computing asymptotic lower and upper bounds on any polynomial expression over the program variables of a Prob-solvable loop. This procedure allows us to derive bounds for $\mathbb{E}(G_{i+1} - G_i \mid \mathcal{F}_i)$ and the branches of G . Using the procedure of Section 4.2, we are able to automate the proof rules from Chapter 3 for certifying termination and non-termination of Prob-solvable loops.

Before formalizing our procedure and its required notions, the following examples illustrate how reasoning with asymptotic bounds helps to apply termination and non-termination proof rules to Prob-solvable loops.

Example 17 (Bounds & RSM-Rule). *Consider the following Prob-solvable loop:*

```

x := 1
y := 0
while x < 100 do
  | y := y + 1
  | x := 2x + y^2 [1/2] 1/2x
end
  
```

Observe $y_i = i$. The martingale expression for $G = -x + 100$ is $\mathbb{E}(G_{i+1} - G_i \mid \mathcal{F}_i) = -\frac{x_i}{4} - \frac{i^2}{2} - i - \frac{1}{2}$ and contains the probabilistic variable x :

$$\begin{aligned} \mathbb{E}(G_{i+1} - G_i \mid \mathcal{F}_i) &= \frac{1}{2}(100 - 2x_i - (i+1)^2) + \frac{1}{2}(100 - \frac{1}{2}x_i) - (100 - x_i) \\ &= -\frac{x_i}{4} - \frac{i^2}{2} - i - \frac{1}{2} \end{aligned}$$

Note that if the term $-\frac{x_i}{4}$ was not present in $\mathbb{E}(G_{i+1} - G_i \mid \mathcal{F}_i)$, we could immediately certify the program to be PAST using the RSM-Rule because $-\frac{i^2}{2} - i - \frac{1}{2} \leq -\frac{1}{2}$ for all $i \geq 0$.

However, by taking a closer look at the variable x we observe that it is almost-surely lower bounded by some function $\alpha \cdot 2^{-i}$ for some $\alpha \in \mathbb{R}^+$. Therefore, $-\frac{x_i}{4} \leq -\beta \cdot 2^{-i}$ for some $\beta \in \mathbb{R}^+$. From this follows that eventually $\mathbb{E}(G_{i+1} - G_i \mid \mathcal{F}_i) \leq -\gamma \cdot i^2$ for some $\gamma \in \mathbb{R}^+$. This means, G is eventually a RSM and the program is PAST by the RSM-Rule.

Remark 5. For Example 17, it would in fact be enough to impose a lower bound of 0 on x . However, for more complex examples, tight asymptotic bounds are needed to reach the desired result.

Example 18 (Bounds & GR-AST-Rule). Consider the following Prob-solvable loop:

$x := 0$

$y := 1$

$z := 5$

while $z > 0$ **do**

$x := x + 1$
$y := 2y + x^2 \quad [\frac{1}{2}] \quad 2y + 3x$
$z := z + y \quad [\frac{2}{3}] \quad z - y$

end

The martingale expression for $-G = -z$ is $\mathbb{E}(G_i - G_{i+1} \mid \mathcal{F}_i) = -\frac{2y_i}{3} - \frac{i^2}{6} - \frac{5i}{6} - \frac{2}{3}$.

We observe that almost-surely y_i is eventually lower bounded (and upper bounded) by a function $\alpha \cdot 2^i$ for some $\alpha \in \mathbb{R}^+$. Therefore, eventually $\mathbb{E}(G_i - G_{i+1} \mid \mathcal{F}_i) \leq -\beta 2^i$ holds for some $\beta \in \mathbb{R}^+$. Consequently, $-z$ is eventually an ϵ_i -repulsing supermartingale with $\epsilon_i \in \Theta(2^i)$.

Bounds c_i on the differences $|z_{i+1} - z_i|$ can be established by a similar style of reasoning: We have $|z_{i+1} - z_i| = |y_{i+1}| = y_{i+1}$ almost-surely. Moreover, y_{i+1} is eventually upper bounded by a function $\alpha \cdot 2^i$ for some $\alpha \in \mathbb{R}^+$. Hence, eventually $|z_{i+1} - z_i| \leq c_i$ almost-surely for some $c_i \in \Theta(2^i)$. Therefore, also the condition $c_i \in O(\epsilon_i)$ of the GR-AST-Rule is satisfied.

Because all the conditions of the GR-AST-Rule are satisfied and there is a positive probability of reaching any iteration (necessary for the relaxation of the GR-AST-Rule), we conclude that the program is not AST.

Example 19 (Bounds & SM-Rule). Consider the following Prob-solvable loop:

```

x := 10
y := 0
while x > 0 do
  y := y + 1  $\frac{1}{3}$  y + 2  $\frac{1}{3}$  y + 3
  x := x + y2 - 1  $\frac{1}{2}$  x - y2 + 1
end

```

The martingale expression for $G = x$ is $\mathbb{E}(G_{i+1} - G_i \mid \mathcal{F}_i) = 0$, which means x is a martingale. In order to be able to apply the SM-Rule and certify AST for the program, we need to provide a $p \in (0, 1]$ and a $d \in \mathbb{R}^+$ such that eventually x decreases by at least d with a probability of at least p . For that, we consider all branches $B \in \text{supp}(\mathcal{U}_{\mathcal{L}}^G)$. In this example we have $\text{supp}(\mathcal{U}_{\mathcal{L}}^G) = \{x_i + y_i^2 + 2y_i, x_i + y_i^2 + 4y_i + 3, x_i + y_i^2 + 6y_i + 8, x_i - y_i^2 - 2y_i, x_i - y_i^2 - 4y_i - 3, x_i - y_i^2 - 6y_i - 8\}$. All branches occur with the same probability of $\frac{1}{6}$.

Considering the branch $x_i - y_i^2 - 2y_i$, we get that x changes by $-y_i^2 - 2y_i$ in this case. Moreover, y_i is eventually lower bounded (and upper bounded) by some function $\alpha \cdot i$ for some $\alpha \in \mathbb{R}^+$. This leads to the fact that eventually $-y_i^2 - 2y_i \leq -\beta \cdot i^2$ for some $\beta \in \mathbb{R}^+$. Hence, it holds that eventually x decreases by at least 1 with a probability of at least $\frac{1}{6}$. Therefore, we conclude the program to be AST by the SM-Rule.

4.1 Prob-solvable Loops and Monomials

In Section 4.2 we state a procedure computing bounds on polynomial expressions over program variables of a Prob-solvable loop. The procedure computes bounds on monomials of program variables. Because every polynomial is a linear combination of monomials, the procedure can be used to retrieve bounds on any polynomial expressions by replacing every monomial by its upper bound or lower bound depending on the sign of the monomial's coefficient.

For the termination of the proposed procedure, it is important that there are no circular dependencies among monomials. By the definition of Prob-solvable loops, this fact holds for program variables (monomials of order 1). Every Prob-solvable loop \mathcal{L} comes with an ordering on its variables and every variable is restricted to only depend linearly on itself or polynomially on previous variables. In this section, we prove that the fact of acyclic dependency extends naturally to monomials over program variables.

Definition 19 (Monomial Ordering). Let \mathcal{L} be a Prob-solvable loop with variables x_1, \dots, x_m . Let $y_1 = \prod_{j=1}^m x_j^{p_j}$ and $y_2 = \prod_{j=1}^m x_j^{q_j}$, where $p_j, q_j \in \mathbb{N}$, be two monomials

over the program variables. We define an order \preceq on monomials over the program variables of \mathcal{L} by

$$y_1 \preceq y_2 \iff (p_m, \dots, p_1) \leq_{lex} (q_m, \dots, q_1) \quad (4.1)$$

where \leq_{lex} is the lexicographic order on \mathbb{N}^m . The order \preceq is total because \leq_{lex} is total. With $y_1 \prec y_2$ we denote $y_1 \preceq y_2 \wedge y_1 \neq y_2$.

Example 20 (Monomials). Let \mathcal{L} be a Prob-solvable loop with variables x_1, \dots, x_m . The following statements hold for the monomial order \preceq :

- $1 \prec x_1 \prec x_2 \prec \dots \prec x_{m-1} \prec x_m$
- $x_1^k \prec x_2$ for any $k \in \mathbb{N}$
- $x_1^2 \prec x_1^3$
- $x_3^4 x_2^{100} x_1^{99} \prec x_3^5 x_2^2 x_1^3$

Before proving acyclic dependencies for monomials we establish the following fact.

Lemma 5. Let y_1, y_2, z_1, z_2 be monomials over the program variables of a Prob-solvable loop \mathcal{L} . If $y_1 \preceq z_1$ and $y_2 \preceq z_2$ then $y_1 \cdot y_2 \preceq z_1 \cdot z_2$.

Proof. Let (a_m, \dots, a_1) , (b_m, \dots, b_1) , (c_m, \dots, c_1) and (d_m, \dots, d_1) be the exponents of y_1, y_2, z_1 and z_2 respectively.

Case 1 — $y_1 = z_1$ and $y_2 = z_2$: trivial

Case 2 — $y_1 \prec z_1$ and $y_2 = z_2$:

There is a $j \in \{1, \dots, m\}$ such that $a_j < c_j$ and $a_l = c_l$ for all $l > j$. Therefore, $a_j + b_j < c_j + d_j$ and $a_l + b_l = c_l + d_l$ for all $l > j$, which means $y_1 \cdot y_2 \prec z_1 \cdot z_2$.

Case 3 — $y_1 = z_1$ and $y_2 \prec z_2$: symmetric to Case 2

Case 4 — $y_1 \prec z_1$ and $y_2 \prec z_2$:

There is a $j \in \{1, \dots, m\}$ such that $a_j < c_j$ and $a_l = c_l$ for all $l > j$. Moreover, there is a $k \in \{1, \dots, m\}$ such that $b_k < d_k$ and $b_l = d_l$ for all $l > k$. W.l.o.g. let $j \geq k$. Then, $a_j + b_j < c_j + d_j$ and $a_l + b_l = c_l + d_l$ for all $l > j$, which means $y_1 \cdot y_2 \prec z_1 \cdot z_2$. \square

Using Lemma 5, we now proceed justifying the fact that cyclic dependencies among monomials over program variables of Prob-solvable loops cannot happen.

Lemma 6. (Monomial Acyclic Dependency) Let x be a monomial over the program variables of a Prob-solvable loop \mathcal{L} . For every branch $B \in \text{supp}(\mathcal{U}_{\mathcal{L}}^x)$ and every monomial y in B it holds that $y \preceq x$.

Proof. The proof is by structural induction over monomials. The base case for which x is a single variable holds by the definition of Prob-solvable loops.

Let $x := s \cdot t$ where s and t are monomials over the variables of \mathcal{L} and

- for every $B_s \in \text{supp}(\mathcal{U}_{\mathcal{L}}^s)$ and every monomial u in B_s it holds that $u \preceq s$,
- for every $B_t \in \text{supp}(\mathcal{U}_{\mathcal{L}}^t)$ and every monomial w in B_t it holds that $w \preceq t$,

Let $B \in \text{supp}(\mathcal{U}_{\mathcal{L}}^x)$ be an arbitrary branch of x . By definition of $\mathcal{U}_{\mathcal{L}}^x$ we get $B = B_s \cdot B_t$, where B_s is a branch of s and B_t is a branch of t . Note that B_s and B_t are polynomials over program variables or equivalently linear combinations of monomials. Therefore, for every monomial y in B we have $y = u \cdot w$ where u is a monomial in B_s and w a monomial in B_t .

By the induction hypothesis, $u \preceq s$ and $w \preceq t$. Using Lemma 5, we get $u \cdot w \preceq s \cdot t$ which means $y \preceq x$. \square

Lemma 6 basically states that the value of a monomial x over the program variables of \mathcal{L} only depends on the value of monomials y which precede x in the monomial ordering \preceq . This ensures the dependencies among monomials to be acyclic, as for a cycle to exist there would have to be a dependency of a monomial x to a monomial succeeding x .

For every monomial x , every branch $B \in \text{supp}(\mathcal{U}_{\mathcal{L}}^x)$ is a polynomial over the program variables. With $\text{Rec}(x)$ we denote the set of coefficients of the monomial x in all branches of a Prob-solvable loop. That means, $\text{Rec}(x) := \{\text{coefficient of } x \text{ in } B \mid B \in \text{supp}(\mathcal{U}_{\mathcal{L}}^x)\}$. With $\text{Inhom}(x)$ we denote all the branches of the monomial x without x and its coefficient. That means, $\text{Inhom}(x) := \{B - c \cdot x \mid B \in \text{supp}(\mathcal{U}_{\mathcal{L}}^x) \text{ and } c = \text{coefficient of } x \text{ in } B\}$.

4.2 Asymptotic Bounds for Prob-solvable Loops

Throughout this section, all arising recurrence relations are *C-finite* [KP11] [BKS19]. Closed-form solutions of C-finite recurrence relations are called *C-finite expressions*. C-finite expression can be characterized as *exponential polynomials*, that means sums of products of polynomials with exponential functions $\sum_j p_j(x) \cdot c_j^x$. Therefore, for any C-finite expression f and any constant $r \in \mathbb{R}$, the following important property holds:

$$\exists \alpha, \beta \in \mathbb{R}^+, \exists i_0 \in \mathbb{N} : \forall i \geq i_0 : \alpha f(i) \leq f(i+r) \leq \beta f(i) \quad (4.2)$$

Intuitively, the property states that constant shifts do not change the asymptotic behavior of f . We make use of this property at various proof steps in this section. Another essential property of C-finite recurrences is that their closed-form always exists and can be computed [KP11].

The standard Big-O notation does not differentiate between positive and negative functions, as it considers the absolute value of functions. We, however, need to differentiate between functions like 2^i and -2^i . Therefore, we introduce the following notions.

Definition 20 (Domination). *Let F be a finite set of functions from \mathbb{N} to \mathbb{R} .*

A function $g : \mathbb{N} \rightarrow \mathbb{R}$ is dominating F if eventually $\alpha \cdot g(i) \geq f(i)$ for all $f \in F$ and some $\alpha \in \mathbb{R}^+$.

A function $g : \mathbb{N} \rightarrow \mathbb{R}$ is dominated by F if all $f \in F$ dominate $\{g\}$.

Definition 21 (Bounding Function). *Let \mathcal{L} be a Prob-solvable loop and X an arithmetic expression over the program variables of \mathcal{L} .*

A lower bounding function for X is a monotone function $l : \mathbb{N} \rightarrow \mathbb{R}$ such that eventually $\mathbb{P}(\alpha \cdot l(i) \leq X_i \mid T^{-\mathcal{G}} > i) = 1$ for some $\alpha \in \mathbb{R}^+$. Moreover, l is assumed to be non-negative or non-positive.

An upper bounding function for X is a function $u : \mathbb{N} \rightarrow \mathbb{R}$ such that eventually $\mathbb{P}(X_i \leq \alpha \cdot u(i) \mid T^{-\mathcal{G}} > i) = 1$ for some $\alpha \in \mathbb{R}^+$. Moreover, u is assumed to be non-negative or non-positive.

An absolute bounding function for X is an upper bounding function for $|X|$.

We define $\mathbb{P}(\cdot \mid A) = 1$ if $\mathbb{P}(A) = 0$.

Intuitively, a function f dominates a function g if f eventually surpasses g in its value modulo a positive constant factor. For a finite set F of exponential polynomials, a function dominating F and a function dominated by F are easily computable with standard techniques, by analyzing the terms of the functions in the finite set F . With $\text{dominating}(F)$ we denote an algorithm computing an exponential polynomial dominating F . With $\text{dominated}(F)$ we denote an algorithm computing an exponential polynomial dominated by F . Moreover, we assume the functions returned by the algorithms $\text{dominating}(F)$ and $\text{dominated}(F)$ to be monotone and either non-negative or non-positive.

Example 21 (Domination). *The following statements are true:*

- 0 dominates $\{-i^3 + i^2 + 5\}$
- i^2 dominates $\{2i^2\}$
- $i^2 \cdot 2^i$ dominates $\{i^2 \cdot 2^i + i^9, i^5 + i^3, 2^{-i}\}$
- i is dominated by $\{i^2 - 2i + 1, \frac{1}{2}i - 5\}$
- -2^i is dominated by $\{2^i - i^2, -10 \cdot 2^{-i}\}$

A bounding function, either lower, upper, or absolute, imposes a bound on an expression X over the program variables holding eventually, almost-surely, and modulo a positive constant factor. Moreover, bounds on X only need to hold as long as the program has not terminated.

Given a Prob-solvable loop \mathcal{L} and a monomial x over the program variables of \mathcal{L} , we propose with Algorithm 4.1 a procedure computing a lower and upper bounding function for x . Because every polynomial expression is a linear combination of monomials, the procedure can be used to compute lower and upper bounding functions for any polynomial expression over \mathcal{L} 's program variables by substituting every monomial with its lower or upper bounding function depending on the sign of the monomial's coefficient. Once a lower bounding function $l(i)$ and an upper bounding function $u(i)$ are computed, an absolute bounding function can be computed by $\text{dominating}(\{u(i), -l(i)\})$.

With Algorithm 4.1, we propose a procedure for computing bounding functions for monomials. The symbols c_1 , c_2 and d are symbolic constants representing elements in \mathbb{R}^+ . $\hat{Sign}(x)$ is an over-approximation of the sign of the monomial x , that means, if $\exists i : \mathbb{P}(x_i > 0) > 0$ then $+$ $\in \hat{Sign}(x)$ and if $\exists i : \mathbb{P}(x_i < 0) > 0$ then $- \in \hat{Sign}(x)$.

Algorithm 4.1: Computing bounding functions for monomials

Input: A Prob-solvable loop \mathcal{L} and a monomial x over \mathcal{L} 's variables

Output: Lower and upper bounding functions $l(i)$, $u(i)$ for x

- 1 $\text{inhomBoundsUpper} := \{\text{upper bounding function of } P \mid P \in \text{Inhom}(x)\}$
 - 2 $\text{inhomBoundsLower} := \{\text{lower bounding function of } P \mid P \in \text{Inhom}(x)\}$
 - 3 $U(i) := \text{dominating}(\text{inhomBoundsUpper})$
 - 4 $L(i) := \text{dominated}(\text{inhomBoundsLower})$
 - 5 $\text{maxRec} := \max \text{Rec}(x)$
 - 6 $\text{minRec} := \min \text{Rec}(x)$
 - 7 $I := \emptyset$
 - 8 **if** $+$ $\in \hat{Sign}(x)$ **then**
 - 9 | $I := I \cup \{c_1\}$
 - 10 **end**
 - 11 **if** $- \in \hat{Sign}(x)$ **then**
 - 12 | $I := I \cup \{-c_2\}$
 - 13 **end**
 - 14 $\text{upperCandidates} := \text{closed-forms of recurrences}$
 $\{y_{i+1} = r \cdot y_i + d \cdot U(i) \mid r \in \{\text{minRec}, \text{maxRec}\}, y_0 \in I\}$
 - 15 $\text{lowerCandidates} := \text{closed-forms of recurrences}$
 $\{y_{i+1} = r \cdot y_i + d \cdot L(i) \mid r \in \{\text{minRec}, \text{maxRec}\}, y_0 \in I\}$
 - 16 $u(i) := \text{dominating}(\text{upperCandidates})$
 - 17 $l(i) := \text{dominated}(\text{lowerCandidates})$
 - 18 **return** $l(i), u(i)$
-

Lemma 6 states that for Prob-solvable loops there are no cyclic dependencies among

monomials. Lemma 6, the computability of closed forms of C-finite recurrences and the fact that within a Prob-solvable loop there are only finitely many monomials present, implies the termination of Algorithm 4.1. The correctness is stated in the next theorem.

Theorem 11 (Correctness of Algorithm 4.1). *The functions $l(i), u(i)$ returned by Algorithm 4.1 on input \mathcal{L} and x are bounding functions for x , where $l(i)$ is a lower bounding function and $u(i)$ is an upper bounding function.*

Proof. Intuitively, we have to show that regardless of the paths through the loop body taken by any program run, the value of x is always eventually upper bounded by some function in *upperCandidates* and eventually lower bounded by some function in *lowerCandidates* (almost-surely and modulo constant factors). We only show that x is always eventually upper bounded by some function in *upperCandidates*. The proof for the lower bounding function is analogous.

Let $\vartheta \in \Sigma$ be a *possible* program run, that means $\mathbb{P}(\text{Cyl}(\pi)) > 0$ for all finite prefixes π of ϑ . Then, for every $i \in \mathbb{N}$, if $T^{-\mathcal{G}}(\vartheta) > i$, the following holds:

$$\begin{aligned} x_{i+1}(\vartheta) &= a_{(1)} \cdot x_i(\vartheta) + P_{(1)i}(\vartheta) \\ &\text{or} \\ x_{i+1}(\vartheta) &= a_{(2)} \cdot x_i(\vartheta) + P_{(2)i}(\vartheta) \\ &\text{or ... or} \\ x_{i+1}(\vartheta) &= a_{(k)} \cdot x_i(\vartheta) + P_{(k)i}(\vartheta), \end{aligned}$$

where $a_{(j)} \in \text{Rec}(x)$ and $P_{(j)} \in \text{Inhom}(x)$ are polynomials over program variables or equivalently, linear combinations of monomials.

Let $u_1(i), \dots, u_k(i)$ be upper bounding functions of $P_{(1)}, \dots, P_{(k)}$, which are computed recursively at line 14. Moreover, let $U(i) := \text{dominating}(\{u_1(i), \dots, u_k(i)\})$, $\text{minRec} = \min \text{Rec}(x)$ and $\text{maxRec} = \max \text{Rec}(x)$.

Let $l_0 \in \mathbb{N}$ be the smallest number such that for all $j \in \{1, \dots, k\}$ and $i \geq l_0$:

$$\mathbb{P}(P_{(j)i} \leq \alpha_j \cdot u_j(i) \mid T^{-\mathcal{G}} > i) = 1 \text{ for some } \alpha_j \in \mathbb{R}^+, \text{ and} \quad (4.3)$$

$$u_j(i) \leq \beta \cdot U(i) \text{ for some } \beta \in \mathbb{R}^+ \quad (4.4)$$

That means, all inequalities from the bounding functions u_j and the dominating function U hold after l_0 .

Because U is a dominating function, it is by definition either non-negative or non-positive. Assume $U(i)$ to be non-negative, the case for which $U(i)$ is non-positive is symmetric.

Using the facts (4.3) and (4.4), we establish the following: For the constant $\gamma := \beta \cdot \max_{j=1..k} \alpha_j$ it holds that $\mathbb{P}(P_{(j)i} \leq \gamma \cdot U(i) \mid T^{-\mathcal{G}} > i) = 1$ for all $j \in \{1, \dots, k\}$ and all $i \geq l_0$.

Let l_1 be the smallest number such that $l_1 \geq l_0$ and $U(i + l_0) \leq \delta \cdot U(i)$ for all $i \geq l_1$ and some $\delta \in \mathbb{R}^+$.

Case 1 — x_i is almost-surely negative for all $i \geq l_1$:

Consider the recurrence relation $y_0 = m$, $y_{i+1} = \min \text{Rec} \cdot y_i + \eta \cdot U(i)$, where $\eta := \max(\gamma, \delta)$ and m is the maximum value of $x_{l_1}(\vartheta)$ among all possible program runs ϑ . Note that m exists because there are only finitely many values $x_{l_1}(\vartheta)$ for possible program runs ϑ . Moreover, m is negative by our case assumption.

By induction, we get $\mathbb{P}(x_i \leq y_{i-l_1} \mid T^{-\mathcal{G}} > i) = 1$ for all $i \geq l_1$.

Therefore, for a closed-form solution $s(i)$ to the recurrence relation y_i , we get $\mathbb{P}(x_i \leq s(i - l_1) \mid T^{-\mathcal{G}} > i) = 1$ for all $i \geq l_1$. We emphasize that s exists and can effectively be computed because y_i is C-finite. Moreover, $s(i - l_1) \leq \theta \cdot s(i)$ for all $i \geq l_2$ for some $l_2 \geq l_1$ and some $\theta \in \mathbb{R}^+$. Therefore, s satisfies the bound condition of an upper bounding function

Also, s is present in *upperCandidates* by choosing the symbolic constants c_2 and d to represent $-m$ and η respectively.

The function $u(i) := \text{dominating}(\text{upperCandidates})$, at line 16, is dominating *upperCandidates* (hence also s), is monotone and either non-positive or non-negative. Therefore, $u(i)$ is an upper bounding function for x .

Case 2 — x_i is not almost-surely negative for all $i \geq l_1$:

That means, there is a possible program run ϑ' such that $x_i(\vartheta') \geq 0$ for some $i \geq l_1$. Let $l_2 \geq l_1$ be the smallest number such that $x_{l_2}(\hat{\vartheta}) \geq 0$ for some possible program run $\hat{\vartheta}$. This number certainly exists, as $x_i(\vartheta')$ is non-negative for some $i \geq l_1$.

Consider the recurrence relation $y_0 = m$, $y_{i+1} = \max \text{Rec} \cdot y_i + \eta \cdot U(i)$, where $\eta := \max(\gamma, \delta)$ and m is the maximum value of $x_{l_2}(\vartheta)$ among all possible program runs ϑ . Note that m exists because there are only finitely many values $x_{l_2}(\vartheta)$ for possible program runs ϑ . Moreover, m is non-negative because $m \geq x_{l_2}(\hat{\vartheta}) \geq 0$.

By induction, we get $\mathbb{P}(x_i \leq y_{i-l_2} \mid T^{-\mathcal{G}} > i) = 1$ for all $i \geq l_2$.

Therefore, for a solution $s(i)$ to the recurrence relation y_i , we get $\mathbb{P}(x_i \leq s(i - l_2) \mid T^{-\mathcal{G}} > i) = 1$ for all $i \geq l_2$. We again emphasize that s exists and can effectively be computed because y_i is C-finite. Moreover, $s(i - l_2) \leq \theta \cdot s(i)$ for all $i \geq l_3$ for some $l_3 \geq l_2$ and some $\theta \in \mathbb{R}^+$. Therefore, s satisfies the bound condition of an upper bounding function

Also, s is present in *upperCandidates* by choosing the symbolic constants c_1 and d to represent m and η respectively.

The function $u(i) := \text{dominating}(\text{upperCandidates})$, at line 16, is dominating *upperCandidates* (hence also s), is monotone and either non-positive or non-negative. Therefore, $u(i)$ is an upper bounding function for x . \square

Next, we illustrate Algorithm 4.1 by computing bounding functions for x and the Prob-solvable loop from Example 17.

Example 22 (Bounding functions). *Consider again the Prob-solvable loop from Example 17:*

```

 $x := 1$ 
 $y := 0$ 
while  $x < 100$  do
  |  $y := y + 1$ 
  |  $x := 2x + y^2 \lfloor \frac{1}{2} \rfloor \frac{1}{2}x$ 
end

```

We have $Rec(x) := \{2, \frac{1}{2}\}$ and $Inhom(x) = \{y^2, 0\}$. Computing bounding functions recursively for $P \in Inhom(x)$ is simple, in this example, as we can give exact bounds for y^2 and 0 leading to $inhomBoundsUpper = \{i^2, 0\}$ and $inhomBoundsLower = \{i^2, 0\}$ on line 1 and 2 of Algorithm 4.1. Consequently, we get

- $U(i) = i^2$,
- $L(i) = 0$,
- $maxRec = 2$ and
- $minRec = \frac{1}{2}$

in the lines 3 to 6.

With a rudimentary static analysis of the loop, we determine the (exact) over-approximation $Sign(x) := \{+\}$ by observing that $x_0 > 0$ and all $P \in Inhom(x)$ are strictly positive. Therefore, $upperCandidates$ is the set of closed-form solutions of the recurrences

- $y_0 := c_1, y_{i+1} := 2y_i + d \cdot i^2$ and
- $y_0 := c_1, y_{i+1} := \frac{1}{2}y_i + d \cdot i^2$.

Similarly, $lowerCandidates$ is the set of closed-form solutions of the recurrences

- $y_0 := c_1, y_{i+1} := 2y_i$ and
- $y_0 := c_1, y_{i+1} := \frac{1}{2}y_i$.

Using any algorithm for computing closed-forms of C-finite recurrences we can determine $upperCandidates = \{c_1 2^i - di^2 - 2di + 3d2^i - 3d, c_1 2^{-i} + 2di^2 - 8di - 12d2^{-i} + 12d\}$ and $lowerCandidates = \{c_1 2^i, c_1 2^{-i}\}$

This leads to the upper bounding function $u(i) = 2^i$ on line 16 and the lower bounding function $l(i) = 2^{-i}$ on line 17.

The bounding functions $l(i)$ and $u(i)$ can be used to compute bounding functions for expressions containing x linearly by replacing x by $l(i)$ or $u(i)$ depending on the sign of the coefficient of x . For instance, eventually and almost-surely the following inequality holds:

$$-\frac{x_i}{4} - \frac{i^2}{2} - i - \frac{1}{2} \leq -\frac{1}{4} \cdot \alpha \cdot 2^{-i} - \frac{i^2}{2} - i - \frac{1}{2}$$

for some $\alpha \in \mathbb{R}^+$. The inequality results from replacing x_i by its lower bounding function. Therefore, eventually and almost-surely we get $-\frac{x_i}{4} - \frac{i^2}{2} - i - \frac{1}{2} \leq -\beta \cdot i^2$ for some $\beta \in \mathbb{R}^+$, which means $-i^2$ is an upper bounding function for the expression $-\frac{x_i}{4} - \frac{i^2}{2} - i - \frac{1}{2}$.

4.3 Algorithms for Proof Rules

With Algorithm 4.1 computing bounding functions for polynomial expressions over program variables at hand, we are now able to formalize our algorithmic approaches automating the termination analysis of Prob-solvable loops using the proof rules from Chapter 3. Given a Prob-solvable loop \mathcal{L} and a polynomial expression E over \mathcal{L} 's variables, we denote with $lbf(E)$, $ubf(E)$ and $abf(E)$ functions computing a lower, upper and absolute bounding function for E respectively. Our algorithmic approach for proving PAST using the RSM-Rule is given in Algorithm 4.2.

Algorithm 4.2: Ranking-Supermartingale-Rule

Input: Prob-solvable loop \mathcal{L}

Output: If *true* then \mathcal{L} with G satisfies the RSM-Rule; hence \mathcal{L} is PAST

- 1 $E := \mathbb{E}(G_{i+1} - G_i \mid \mathcal{F}_i)$
 - 2 $u(i) := ubf(E)$
 - 3 $limit := \lim_{i \rightarrow \infty} u(i)$
 - 4 **if** $limit < 0$ **then**
 - 5 | **return true**
 - 6 **else**
 - 7 | **return false**
 - 8 **end**
-

The following example illustrates Algorithm 4.2 for the Prob-solvable loop from Examples 17 and 22.

Example 23 (Algorithm 4.2). *Consider again the Prob-solvable loop \mathcal{L} from Examples 17 and 22:*

```

x := 1
y := 0
while x < 100 do
  | y := y + 1
  | x := 2x + y2 [1/2] 1/2x
end
    
```

Applying Algorithm 4.2 on \mathcal{L} leads to $E = -\frac{x_i}{4} - \frac{i^2}{2} - i - \frac{1}{2}$ on line 1. On line 2, we get the upper bounding function $u(i) := -i^2$ for E . Because $\lim_{i \rightarrow \infty} u(i) < 0$, Algorithm 4.2 returns true on line 5. This is valid because $u(i)$ having a negative limit witnesses that the martingale expression E eventually decreases by a constant and therefore is a RSM.

Correctness of Algorithm 4.2: When returning true at line 5 we have $\mathbb{P}(E_i \leq \alpha \cdot u(i) \mid T^{-G} > i) = 1$ for all $i \geq i_0$ and some $i_0 \in \mathbb{N}$, $\alpha \in \mathbb{R}^+$. Moreover, $u(i) < -\epsilon$ for all $i \geq i_1$ for some $i_1 \in \mathbb{N}$, by the definition of lim. From this follows that $\forall i \geq \max(i_0, i_1)$ we almost-surely have $\mathcal{G}_i \implies \mathbb{E}(G_{i+1} - G_i \mid \mathcal{F}_i) \leq -\alpha\epsilon$, which means G is eventually a RSM and the program \mathcal{L} is PAST by the RSM-Rule.

Our approach proving AST using the SM-Rule is captured with Algorithm 4.3

Algorithm 4.3: Supermartingale-Rule

Input: Prob-solvable loop \mathcal{L}
Output: If true, \mathcal{L} with G satisfies the SM-Rule with constant d and constant p ;
 hence \mathcal{L} is AST

```

1  $E := \mathbb{E}(G_{i+1} - G_i \mid \mathcal{F}_i)$ 
2  $u(i) := ubf(E)$ 
3 if not eventually  $u(i) \leq 0$  then
4   | return false
5 end
6 for  $B \in \text{supp}(\mathcal{U}_{\mathcal{L}}^G)$  do
7   |  $D := B - G$ 
8   |  $d(i) := ubf(D)$ 
9   |  $limit := \lim_{i \rightarrow \infty} d(i)$ 
10  | if  $limit < 0$  then
11  |   | return true
12  | end
13 end
14 return false
    
```

The following example illustrates Algorithm 4.4 for the Prob-solvable loop from Example 19.

Example 24 (Algorithm 4.3). Consider again the Prob-solvable loop \mathcal{L} from Example 19:

```

 $x := 10$ 

 $y := 0$ 

while  $x > 0$  do
  |  $y := y + 1$   $\left[\frac{1}{3}\right]$   $y + 2$   $\left[\frac{1}{3}\right]$   $y + 3$ 
  |  $x := x + y^2 - 1$   $\left[\frac{1}{2}\right]$   $x - y^2 + 1$ 
end

```

Applying Algorithm 4.3 on \mathcal{L} we get $E \equiv 0$ on line 1 and therefore $u(i) = 0$ on line 2, which means the if-statement on line 3 is not executed.

The expression G ($= x$) has six branches. One of them is $x_i - y_i^2 - 2y_i$, which occurs with probability $\frac{1}{6}$. When the for-loop reaches this branch $B = x_i - y_i^2 - 2y_i$ on line 6, it computes the difference $D = -y_i^2 - 2y_i$ on line 7. An upper bounding function for D on line 8 is given by $d(i) = -i^2$. Because $\lim_{i \rightarrow \infty} d(i) < 0$ Algorithm 4.3 returns true on line 11. This is valid because of the branch B witnessing that G eventually decreases by at least a constant with probability $\frac{1}{6}$. Therefore, all conditions of the SM-Rule are satisfied and \mathcal{L} is AST.

Correctness of Algorithm 4.3: With the same reasoning as for the correctness of Algorithm 4.2, G is a supermartingale if Algorithm 4.3 returns true at line 11. Moreover, there is a branch $B \in \text{supp}(\mathcal{U}_{\mathcal{L}}^G)$ such that G changes eventually and almost-surely by at most $d(i)$. In addition, because $\lim_{i \rightarrow \infty} d(i) < 0$, it follows that $d(i) \leq -\epsilon$ for all $i \geq i_0$ for some $i_0 \in \mathbb{N}$, $\epsilon \in \mathbb{R}^+$. Therefore, eventually G decreases by at least the constant ϵ with probability of at least constant $\mathcal{U}_{\mathcal{L}}^G(B) > 0$. Hence, by the SM-Rule, the input program is AST.

As established in Section 3.5, the relaxation of the GR-AST-Rule requires that there is a positive probability of reaching the iteration i_0 after which the conditions of the proof rule hold. Regarding automation, we strengthen the condition to: There is a positive probability of reaching any iteration, that means $\forall i \in \mathbb{N} : \mathbb{P}(T^{-G} > i) > 0$. Obviously, from $\forall i \in \mathbb{N} : \mathbb{P}(T^{-G} > i) > 0$ follows $\mathbb{P}(T^{-G} > i_0) > 0$. Furthermore, with $\check{\text{CanReachAnyIteration}}(\mathcal{L})$ we denote a computable under-approximation of $\forall i \in \mathbb{N} : \mathbb{P}(T^{-G} > i) > 0$. That means, $\check{\text{CanReachAnyIteration}}(\mathcal{L})$ implies $\forall i \in \mathbb{N} : \mathbb{P}(T^{-G} > i) > 0$. Our approach proving non-AST is summarized in Algorithm 4.4.

The following example illustrates Algorithm 4.4 for the Prob-solvable loop from Example 18.

Example 25 (Algorithm 4.4). Consider again the Prob-solvable loop \mathcal{L} from Example 18:

Algorithm 4.4: Generalized-Repulsing-AST-Rule

Input: Prob-solvable loop \mathcal{L}
Output: if *true*, \mathcal{L} with $-G$ satisfies the GR-AST-Rule; hence \mathcal{L} is not AST

- 1 $E := \mathbb{E}(-G_{i+1} + G_i \mid \mathcal{F}_i)$
- 2 $u(i) := \text{ubf}(E)$
- 3 **if** *not eventually* $u(i) \leq 0$ **then**
- 4 | **return** false
- 5 **end**
- 6 **if** $\neg \check{\text{CanReachAnyIteration}}(\mathcal{L})$ **then**
- 7 | **return** false
- 8 **end**
- 9 $\epsilon(i) := -u(i)$
- 10 $\text{differences} := \{B + G \mid B \in \text{supp}(\mathcal{U}_{\mathcal{L}}^{-G})\}$
- 11 $\text{diffBounds} := \{\text{abf}(d) \mid d \in \text{differences}\}$
- 12 $c(i) := \text{dominating}(\text{diffBounds})$
- 13 **if** $c(i) \in O(\epsilon(i))$ **then**
- 14 | **return** true
- 15 **else**
- 16 | **return** false
- 17 **end**

$$x := 0$$

$$y := 1$$

$$z := 5$$
while $z > 0$ **do**

$x := x + 1$
$y := 2y + x^2 \quad [\frac{1}{2}] \quad 2y + 3x$
$z := z + y \quad [\frac{2}{3}] \quad z - y$

end

Applying Algorithm 4.4 on \mathcal{L} leads to $E = -\frac{2yi}{3} - \frac{i^2}{6} - \frac{5i}{6} - \frac{2}{3}$ on line 1 and to the upper bounding function $u(i) = -2^i$ for E on line 2. Therefore, the if-statement on line 3 is not executed, which means $-G$ is eventually a repulsing supermartingale. Moreover, with a simple static analysis of the loop, we establish $\check{\text{CanReachAnyIteration}}(\mathcal{L})$ to be true, as there is always a positive probability to increase the loop guard. This means, also the if-statement on line 6 is not executed.

E eventually decreases by $\epsilon(i) = 2^i$ (modulo a constant factor), because $u(i) = -2^i$ is an upper bounding function for E . The sets *differences* and *diffBounds* on the lines 10 and 11 contain the following expressions representing all possibilities of $-G_{i+1} + G_i$ and

their corresponding absolute bounding functions:

differences E	$diffBounds$
$x_i^2 + 2x_i + 2y_i + 1$	2^i
$3x_i + 2y_i + 3$	2^i
$-x_i^2 - 2x_i - 2y_i - 1$	2^i
$-3x_i - 2y_i - 3$	2^i

As a result on line 12 we have $c(i) = 2^i$, which eventually and almost-surely is an upper bound on $|-G_{i+1} + G_i|$ (modulo a constant factor). Therefore, also $c(i) \in O(\epsilon(i))$ and the algorithm returns true on line 14. This is correct, as all the preconditions of the GR-AST-Rule are satisfied and therefore \mathcal{L} is not AST.

Correctness of Algorithm 4.4: With the same reasoning as for the correctness of Algorithm 4.3, $-G$ is a supermartingale if Algorithm 4.4 returns true at line 14. Moreover, $-G_0 < 0$ (assuming \mathcal{L} does not terminate in its initial state) and if \mathcal{L} terminates $-G \geq 0$. Therefore, $-G$ eventually is an ϵ_i -repulsing supermartingale. The sequence $(\epsilon_i)_{i \in \mathbb{N}}$ is given by $\epsilon_i := -u(i)$ because $u(i)$ is an upper bounding function for $\mathbb{E}(-G_{i+1} + G_i | \mathcal{F}_i)$. The function $c(i)$, assigned to $dominating(diffBounds)$ on line 12, is a function dominating absolute bounding functions of all branches of $-G_{i+1} + G_i$. Consequently, $c(i)$ is a bound on the differences of G , meaning for $c_i := c(i)$, we have $|-G_{i+1} + G_i| \leq c_i$. Hence, at line 14 additionally $c_i \in O(\epsilon_i)$ holds. Thus, all preconditions of the GR-AST-Rule are satisfied and \mathcal{L} is not AST if Algorithm 4.4 returns true at line 14.

We finally provide Algorithm 4.5 for the R-PAST-Rule. The algorithm is a variation of Algorithm 4.4 (for the GR-AST-Rule). The if-statement on line 2 forces $-G$ to be a martingale. Therefore, after the if-statement $-G$ is an ϵ_i -repulsing supermartingale with $(\epsilon_i)_{i \in \mathbb{N}} \equiv 0$. Moreover, the condition on line 8 ensures, if the algorithm returns true, the absolute differences of $-G$ to be bounded by a constant.

4.4 Rule out Rules

A question arising when combining the algorithms of Section 4.3 into a single procedure is, given a Prob-solvable loop \mathcal{L} , what algorithm to apply first for determining \mathcal{L} 's termination behavior. In [BKS19] the authors provide an algorithm for computing an algebraically closed-form of $\mathbb{E}(M_i)$ for a Prob-solvable loop \mathcal{L} , where M is a polynomial over \mathcal{L} 's variables. The following simple lemma explains how the expression $\mathbb{E}(M_{i+1} - M_i)$ relates to the expression $\mathbb{E}(M_{i+1} - M_i | \mathcal{F}_i)$.

Lemma 7 (Rule out Rules). *Let $(M_i)_{i \in \mathbb{N}}$ be a stochastic process. If $\mathbb{E}(M_{i+1} - M_i | \mathcal{F}_i) \leq -\epsilon$ then $\mathbb{E}(M_{i+1} - M_i) \leq -\epsilon$, for any $\epsilon \in \mathbb{R}^+$.*

Algorithm 4.5: Repulsing-PAST-Rule

Input: Prob-solvable loop \mathcal{L}
Output: If *true*, \mathcal{L} with $-G$ satisfies the R-PAST-Rule; hence \mathcal{L} is not PAST

```

1  $E := \mathbb{E}(-G_{i+1} + G_i \mid \mathcal{F}_i)$ 
2 if  $E \neq 0$  then
3   | return false
4 end
5  $diffBound$ s :=  $\{B + G \mid B \in \text{supp}(\mathcal{U}_{\mathcal{L}}^{-G})\}$ 
6  $diff$ Bounds :=  $\{abf(d) \mid d \in diffBound$ s $\}$ 
7  $c(i) := \text{dominating}(diff$ Bounds)
8 if  $c(i) \in O(1)$  then
9   | return true
10 else
11   | return false
12 end

```

Proof.

$$\begin{aligned}
\mathbb{E}(M_{i+1} - M_i \mid \mathcal{F}_i) \leq -\epsilon & \implies & \text{(Monotonicity of } \mathbb{E} \text{)} \\
\mathbb{E}(\mathbb{E}(M_{i+1} - M_i \mid \mathcal{F}_i)) \leq \mathbb{E}(-\epsilon) & \iff & \text{(Property of } \mathbb{E}(\cdot \mid \mathcal{F}_i) \text{)} \\
\mathbb{E}(M_{i+1} - M_i) \leq \mathbb{E}(-\epsilon) & \iff & \text{(-}\epsilon \text{ is constant)} \\
\mathbb{E}(M_{i+1} - M_i) \leq -\epsilon & &
\end{aligned}$$

□

The contrapositive of Lemma 7 provides a criterion to rule out the viability of a given proof rule. For a Prob-solvable loop \mathcal{L} , if $\mathbb{E}(G_{i+1} - G_i) \not\leq 0$ then $\mathbb{E}(G_{i+1} - G_i \mid \mathcal{F}_i) \not\leq 0$, meaning G is not a supermartingale. The expression $\mathbb{E}(G_{i+1} - G_i)$ depends only on i and can be computed by $\mathbb{E}(G_{i+1} - G_i) = \mathbb{E}(G_{i+1}) - \mathbb{E}(G_i)$, where $\mathbb{E}(G_i)$ is computed by the algorithm in [BKS19]. Therefore, in some cases, proof rules can automatically be deemed nonviable, without actually having to apply the proof rule and computing the required bounding functions.

The following example illustrates how Lemma 7 can be used to rule out the GR-AST-Rule.

Example 26. Consider the following Prob-solvable loop:

```

 $x := 0$ 
 $y := 0$ 
while  $x^2 + y^2 < 10$  do
  |  $x := x + 1$   $\left[\frac{1}{2}\right]$   $x - 1$ 
  |  $y := y + x$   $\left[\frac{1}{2}\right]$   $y - x$ 
end

```

We have $G := 10 - x^2 - y^2$. The algorithm from [BKS19] reveal that $\mathbb{E}(-G_{i+1} - G_i) = i + 2$. By Lemma 7, we can conclude that $-G$ cannot eventually satisfy the supermartingale condition $\mathbb{E}(-G_{i+1} - G_i \mid \mathcal{F}_i) \leq 0$. Therefore, $-G$ cannot eventually be an ϵ_i -repulsing supermartingale and it is not viable to apply the GR-AST-Rule. In fact the loop is PAST, provable by the RSM-Rule using Algorithm 4.2.

4.5 Implementation and Evaluation

Implementation We implemented and combined the algorithms from Section 4.2 and Section 4.3 in the new software tool AMBER, to stand for *Asymptotic Martingale Bounds*. AMBER together with all the following benchmarks is available at github.com/mmsbrggr/amber. AMBER uses MORA [BKS19] for computing the first-order moments of program variables and the diofant package¹ as its computer algebra system.

The over-approximation $\hat{Sign}(x)$ of the signs of a monomial x used in Algorithm 4.1, is implemented by a simple static analysis: For a monomial x consisting solely of even powers, we have $\hat{Sign}(x) = \{+\}$. For a general monomial x , if $x_0 \geq 0$ and all monomials on which x depends, together with their associated coefficients are always positive, then $- \notin \hat{Sign}(x)$. For example, if $supp(\mathcal{U}_L^x) = \{x_i + 2y_i - 3z_i, x_i + u_i\}$, then $- \notin \hat{Sign}(x)$ if $x_0 \geq 0$ as well as $- \notin \hat{Sign}(y)$, $+ \notin \hat{Sign}(z)$ and $- \notin \hat{Sign}(u)$. Otherwise, $- \in \hat{Sign}(x)$. The over-approximation for $+ \notin \hat{Sign}(x)$ is analogous.

In addition to Algorithm 4.1, which computes bounding functions for monomials of program variables, we implemented the following improvements in AMBER.

1. A monomial x is deterministic, which means it is independent of probabilistic choices, if x has a single branch and only depends on monomials having single branches. In this case, the exact value of x in any iteration is given by its first-order moments and bounding functions can be obtained by using these exact representations.
2. Bounding functions for an odd power p of a monomial x can be computed by $u(i)^p$ and $l(i)^p$, where $u(i)$ is an upper bounding function for x and $l(i)$ a lower bounding function.

¹github.com/diofant/diofant

In situations where the aforementioned two enhancements apply, they allow for computing the bounding functions faster than Algorithm 4.1.

The under-approximation $\check{CanReachAnyIteration}(\mathcal{L})$, used in Algorithm 4.4, needs to satisfy the property that if $\check{CanReachAnyIteration}(\mathcal{L})$ is true, then there is a positive probability that the loop \mathcal{L} reaches any iteration. In AMBER, we implemented the under-approximation in the following way: $\check{CanReachAnyIteration}(\mathcal{L})$ is true if there is a branch B of the loop guard polynomial $G_{\mathcal{L}}$ such that $B - G_{\mathcal{L}i}$ is non-negative for all $i \in \mathbb{N}$. Otherwise, $\check{CanReachAnyIteration}(\mathcal{L})$ is false. In other words, if $\check{CanReachAnyIteration}(\mathcal{L})$ is true, then in any iteration there is a positive probability of $G_{\mathcal{L}}$ not decreasing. Because the loop guard of \mathcal{L} is equivalent to $G_{\mathcal{L}} > 0$, for any iteration, there is a positive probability of having a next iteration. Now, the property that if $\check{CanReachAnyIteration}(\mathcal{L})$ is true then there is a positive probability that \mathcal{L} reaches any iteration, follows from the Markov property of the loop \mathcal{L} .

Benchmarks We evaluated AMBER against 37 PPs which can be modeled as Prob-solvable loops. These programs have either been introduced in the literature on probabilistic programming [BKS19], [GGH19], [MMKK17], [NCH18], are adaptations of well-known probabilistic processes or have been designed specifically to test unique features of AMBER, like the ability to handle polynomial real arithmetic.

Our benchmarks are separated into the three categories: (1) programs which are PAST (Table 4.1), (2) programs which are AST (Table 4.2) but not necessarily PAST and (3) programs which are not AST (Table 4.3).

Evaluation Concerning programs which are PAST, we compare AMBER against the tool ABSYNTH [NCH18]. ABSYNTH uses a system of inference rules over the syntax of PPs, to derive bounds on the expected resource consumption of a program and can, therefore, be used to certify PAST. In comparison to AMBER, ABSYNTH requires the degree of the bound to be provided upfront. Moreover, ABSYNTH cannot refute the existence of a bound and therefore cannot handle programs that are not PAST. To the best of our knowledge, AMBER is the first tool capable of certifying AST for PPs which are not PAST as well as the first tool able to prove non-AST.

With AMBER-LIGHT we refer to an implementation of AMBER without the relaxations of the proof rules introduced in Section 3.5. That is, with AMBER-LIGHT the conditions of the proof rules need to hold for all $i \in \mathbb{N}$, whereas with AMBER the conditions are allowed to only hold eventually. For all benchmarks, we compare AMBER against AMBER-LIGHT to show the effectiveness of the respective relaxations. Every benchmark has been run on a machine with a 2.2 GHz Intel i7 (Gen 6) processor and 16 GB of RAM and finished within a timeout of 20 seconds.

Our experimental results from Tables 4.1, 4.2 and 4.3 demonstrate the following:

- AMBER outperforms the state-of-the-art in automating PAST certification (Table 4.1).

Program	AMBER	AMBER-LIGHT	ABSINTH
2d_bounded_random_walk	✓	✓	✗
biased_random_walk_constant	✓	✓	✓
biased_random_walk_exp	✓	✓	✗
biased_random_walk_poly	✓	✗	✗
binomial_past	✓	✓	✓
complex_past	✓	✗	✗
consecutive_bernoulli_trails	✓	✓	✓
coupon_collector_2	✓	✗	✗
coupon_collector_4	✓	✗	✗
dueling_cowboys	✓	✓	✓
exponential_past_1	✓	✓	NA
exponential_past_2	✓	✓	NA
geometric	✓	✓	✓
linear_past_1	✓	✓	✗
linear_past_2	✓	✓	✗
polynomial_past_1	✓	✗	✗
polynomial_past_2	✓	✗	✗
Total ✓	17	11	5

Table 4.1: Benchmarks for AMBER and Prob-solvable loops which are PAST. ✓ symbolizes that the respective tool successfully certified PAST for the given program. ✗ means it failed to certify PAST. NA indicates that the program is out-of-scope for the respective tool because real or rational numbers are necessary to model the program.

- Complex PPs which are AST and not PAST as well as PPs which are not AST can automatically be certified as such by AMBER (Tables 4.2 and 4.3).
- The relaxations of the proof rules we introduced in Section 3.5 are crucial in automating the termination analysis of PPs.

Program	AMBER	AMBER-LIGHT
gambling	✓	✓
symmetric_random_walk_constant_1	✓	✓
symmetric_random_walk_constant_2	✓	✓
symmetric_random_walk_exp_1	✓	✗
symmetric_random_walk_exp_2	✓	✗
symmetric_random_walk_linear_1	✓	✗
symmetric_random_walk_linear_2	✓	✓
symmetric_random_walk_poly_1	✓	✗
symmetric_random_walk_poly_2	✓	✗
Total ✓	9	4

Table 4.2: Benchmarks for AMBER and Prob-solvable loops which are AST and not necessarily PAST. ✓ symbolizes that the respective tool successfully certified PAST for the given program. ✗ means it failed to certify PAST.

Program	AMBER	AMBER-LIGHT
biased_random_walk_nast_constant	✓	✓
biased_random_walk_nast_exp	✓	✓
biased_random_walk_nast_linear	✓	✗
biased_random_walk_nast_poly	✓	✗
binomial_nast	✓	✓
exponential_nast_1	✓	✓
exponential_nast_2	✓	✓
linear_nast_1	✓	✗
linear_nast_2	✓	✗
polynomial_nast_1	✓	✗
polynomial_nast_2	✓	✗
Total ✓	11	5

Table 4.3: Benchmarks for AMBER and Prob-solvable loops which are not AST. ✓ symbolizes that the respective tool successfully certified PAST for the given program. ✗ means it failed to certify PAST.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Related Work

Probabilistic Termination Proof Rules

In Chapter 3 of this thesis we stated several proof rules based on (super-)martingales. Martingale based techniques were first recognized to be applicable for verification of PPs and probabilistic termination analysis in [CS13]. The authors introduced the notion of a RSM as well as the RSM-Rule. [FFH15] extended the work of [CS13] to allow for non-determinism and continuous probability distributions. Moreover, [FFH15] showed completeness of the RSM-Rule for their class of PPs.

More recently, [ACN17] advanced RSMs to *Lexicographic Ranking Supermartingales*. The authors determined that for some programs there are no linear RSMs, however, linear lexicographic ranking supermartingales do exist.

The SM-Rule from Chapter 3 was introduced in [MMKK17]. A proof rule similar to the SM-Rule was given in [HFC18]. Both proof rules are based on supermartingales which do not need to be ranking. They both can certify AST for programs that are not necessarily PAST. In [HFC18] the authors confirmed that their proof rule is independent of the SM-Rule.

[TOUH18] examined martingale based techniques for probabilistic reachability from an order-theoretic viewpoint. This led the authors to the notions of *Nonnegative Repulsing Supermartingales* and *γ -Scaled Submartingales*. These notions are accompanied by their own proof rules for which the authors proved soundness and completeness.

The R-AST-Rule, which we generalize in Section 3.4, is able to refute AST and was proposed in [CNZ17]. The paper introduced *Repulsing Supermartingales* not only for refuting AST but mainly for obtaining bounds on the probability of stochastic invariants.

A different method, not based on martingales, for the probabilistic termination analysis of PPs, was given by a weakest-precondition-style calculus in [KKMO16]. The calculus

is defined over the syntax of a probabilistic while-language and can be used to calculate bounds on the expected termination time of a PP.

Automation of Martingale Techniques

[CS13] proposed a procedure for synthesizing linear martingales and supermartingales using *Farkas' Lemma*. *Azuma's Inequality* is then applied to the discovered martingales and supermartingales to obtain concentration results. In [CFNH18] the algorithmic construction of supermartingales was extended to allow for non-determinism and in [CFG16] to polynomial supermartingales.

[CNZ17], which introduced repulsing supermartingales and stochastic invariants, also provided an algorithmic approach for constructing linear repulsing supermartingales for a stochastic invariant. However, the R-AST-Rule, as such, was not automated.

Other Related Work

In [GGH19] the authors developed a procedure for computing the expected runtime of constant probability programs. Moreover, the paper suggested a complete decision procedure for AST and PAST for constant probability programs. In this thesis, in Section 3.6, we discovered a fragment of Prob-solvable loops for which AST can be decided and generalized the corresponding result of [GGH19].

A sound and complete procedure deciding AST for *weakly finite programs* was given in [EGK12]. Weakly finite programs are probabilistic programs such that the number of states reachable from any initial state is finite.

[NCH18] gave an algorithmic approach, based on potential functions, for computing bounds on the expected resource consumption of PPs. The paper is accompanied by the tool ABSYNTH, against which we compared our tool AMBER in Section 4.5.

Finally, the class of Prob-solvable loops was first studied in [BKS19]. The authors provided a procedure for computing moment-based invariants as well as the tool MORA which we use in our tool AMBER.

Conclusion

The question of termination for traditional programs is one of the oldest and most infamous challenges in computer science. The generalization of this question to PPs through the notions of AST and PAST can be conquered with techniques based on martingales.

In chapter 3, we introduced several existing proof rules, involving (super-)martingales, for AST, PAST and their negations. The R-AST-Rule, first studied in [CNZ17] and based on the concept of repulsing supermartingales, can be used to certify non-AST. We generalized the R-AST-Rule in Section 3.4 to allow for repulsing supermartingales without c -bounded differences. This allowed us to witness non-AST for PP with unbounded polynomial updates.

In Section 3.6, we established the fragment of admissible PE-RWs, a class of PPs. We relaxed the previously introduced proof rules and combined them with our generalization of the R-AST-Rule into a decision procedure effectively deciding AST for PE-RWs. Our class of PE-RWs strictly subsumes the class of constant probability programs for which AST was shown to be decidable in [GGH19].

Concerning automation of the aforementioned proof rules for a more general class of PPs, in Chapter 4, the class of Prob-solvable loops proved to be a suitable candidate. Various relevant PPs can be modeled as Prob-solvable loops. Moreover, we established that the structural constraints defining Prob-solvable loops allow for automatically computing almost-sure asymptotic bounds on polynomial expressions over program variables. This fact enabled us to automate various probabilistic termination proof rules and bring them together in the tool AMBER.

Experimental evaluations verified the improvement over the state-of-art provided by AMBER. AMBER is the first tool to automate the SM-Rule and the GR-AST-Rule for any class of PPs and can automatically determine the probabilistic termination behavior of programs which are out of reach for other tools.

Future Work An interesting direction for future research is program transformations which simplify programs while preserving their termination behavior. Specifically, transformations which convert a PP outside of Prob-solvable loops into a Prob-solvable loop, as this would expand the applicability of the automation techniques from Chapter 4. Moreover, transforming PPs into PE-RW, while preserving their termination behavior, allows for enlarging the class of PPs for which AST can be effectively decided.

Acronyms

- AST** almost-surely-terminating. 9, 10, 12, 15, 17–21, 24–29, 31–34, 39, 48–51, 54–57, 59–62
- GR-AST-Rule** Generalized-Repulsing-AST-Rule. 24, 28, 29, 31, 33, 35, 36, 38, 39, 49–53, 61
- MDP** Markov Decision Process. 5, 7
- PAST** positively-almost-surely-terminating. 9, 10, 12, 15–20, 26, 27, 29, 32, 33, 38, 47, 48, 52–57, 59–61
- PE-RW** poly-exponential random-walk. 29–34, 61, 62
- PP** probabilistic program. 1–3, 5, 7, 10, 20, 21, 26, 29, 31, 54, 55, 59–62
- R-AST-Rule** Repulsing-AST-Rule. 18–21, 29, 59–61
- R-PAST-Rule** Repulsing-PAST-Rule. 19, 20, 35, 51, 52
- RSM** ranking-supermartingale. 15–19, 26, 31, 33, 38, 48, 59
- RSM-Rule** Ranking-Supermartingale-Rule. 16, 26, 27, 29, 31, 33, 35, 37, 38, 47, 48, 53, 59
- SM** supermartingale. 12, 13, 15, 17, 18, 20–22, 31
- SM-Rule** Supermartingale-Rule. 17–19, 26, 27, 29, 31, 33, 35, 36, 39, 48, 49, 59, 61



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Bibliography

- [ACN17] Sheshansh Agrawal, Krishnendu Chatterjee, and Petr Novotný. Lexicographic ranking supermartingales: an efficient approach to termination of probabilistic programs. *Proceedings of the ACM on Programming Languages*, 2(POPL):34:1–34:32, December 2017.
- [ARS13] Nimar S. Arora, Stuart J. Russell, and Erik B. Sudderth. NET-VISA: Network Processing Vertically Integrated Seismic Analysis. 2013.
- [BBR⁺15] John E. Bistline, David M. Blum, Chris Rinaldi, Gabriel Shields-Estrada, Siegfried S. Hecker, and Marie-Elisabeth Paté-Cornell. A Bayesian Model to Assess the Size of North Korea’s Uranium Enrichment Program. 2015.
- [BKS19] Ezio Bartocci, Laura Kovács, and Miroslav Stankovič. Automatic Generation of Moment-Based Invariants for Prob-Solvable Loops. *arXiv:1905.02835 [cs]*, May 2019. arXiv: 1905.02835.
- [CFG16] Krishnendu Chatterjee, Hongfei Fu, and Amir Kafshdar Goharshady. Termination Analysis of Probabilistic Programs Through Positivstellensatz’s. In Swarat Chaudhuri and Azadeh Farzan, editors, *Computer Aided Verification*, Lecture Notes in Computer Science, pages 3–22, Cham, 2016. Springer International Publishing.
- [CFNH18] Krishnendu Chatterjee, Hongfei Fu, Petr Novotný, and Rouzbeh Hasheminezhad. Algorithmic Analysis of Qualitative and Quantitative Termination Problems for Affine Probabilistic Programs. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 40(2):7:1–7:45, May 2018.
- [CNZ17] Krishnendu Chatterjee, Petr Novotny, and Dorde Zikelic. Stochastic Invariants for Probabilistic Termination. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, POPL 2017, pages 145–160, New York, NY, USA, 2017. ACM. event-place: Paris, France.
- [CPR06] Byron Cook, Andreas Podelski, and Andrey Rybalchenko. Terminator: Beyond Safety. In Thomas Ball and Robert B. Jones, editors, *Computer*

Aided Verification, Lecture Notes in Computer Science, pages 415–418, Berlin, Heidelberg, 2006. Springer.

- [CS13] Aleksandar Chakarov and Sriram Sankaranarayanan. Probabilistic Program Analysis with Martingales. In *Proceedings of the 25th International Conference on Computer Aided Verification - Volume 8044*, CAV 2013, pages 511–526, New York, NY, USA, 2013. Springer-Verlag New York, Inc. event-place: Saint Petersburg, Russia.
- [EGK12] Javier Esparza, Andreas Gaiser, and Stefan Kiefer. Proving Termination of Probabilistic Programs Using Patterns. In P. Madhusudan and Sanjit A. Seshia, editors, *Computer Aided Verification*, Lecture Notes in Computer Science, pages 123–138, Berlin, Heidelberg, 2012. Springer.
- [FDG⁺19] Daniel J. Fremont, Tommaso Dreossi, Shromona Ghosh, Xiangyu Yue, Alberto L. Sangiovanni-Vincentelli, and Sanjit A. Seshia. Scenic: a language for scenario specification and scene generation. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2019, pages 63–78, Phoenix, AZ, USA, June 2019. Association for Computing Machinery.
- [FFH15] Luis María Ferrer Fioriti and Holger Hermanns. Probabilistic Termination: Soundness, Completeness, and Compositionality. In *Proceedings of the 42Nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '15, pages 489–501, New York, NY, USA, 2015. ACM. event-place: Mumbai, India.
- [GGG⁺01] Geoffrey Grimmett, Geoffrey R. Grimmett, Professor of Mathematical Statistics Geoffrey Grimmett, David Stirzaker, and Mathematical Institute David R. Stirzaker. *Probability and Random Processes*. OUP Oxford, May 2001.
- [GGH19] Jürgen Giesl, Peter Giesl, and Marcel Hark. Computing Expected Runtimes for Constant Probability Programs. *arXiv:1905.09544 [cs]*, September 2019. arXiv: 1905.09544.
- [Gha15] Zoubin Ghahramani. Probabilistic machine learning and artificial intelligence. *Nature*, 521(7553):452–459, May 2015. Number: 7553 Publisher: Nature Publishing Group.
- [GKM12] Friedrich Gretz, Joost-Pieter Katoen, and Annabelle McIver. Operational Versus Weakest Precondition Semantics for the Probabilistic Guarded Command Language. In *2012 Ninth International Conference on Quantitative Evaluation of Systems*, pages 168–177, September 2012.
- [HFC18] Mingzhang Huang, Hongfei Fu, and Krishnendu Chatterjee. New Approaches for Almost-Sure Termination of Probabilistic Programs. In Sukyoung Ryu,

editor, *Programming Languages and Systems*, Lecture Notes in Computer Science, pages 181–201, Cham, 2018. Springer International Publishing.

- [HKGK19] Marcel Hark, Benjamin Lucien Kaminski, Jürgen Giesl, and Joost-Pieter Katoen. Aiming Low Is Harder – Induction for Lower Bounds in Probabilistic Program Verification. *arXiv:1904.01117 [cs]*, November 2019. arXiv: 1904.01117.
- [KK15] Benjamin Lucien Kaminski and Joost-Pieter Katoen. On the Hardness of Almost-Sure Termination. In Giuseppe F Italiano, Giovanni Pighizzini, and Donald T. Sannella, editors, *Mathematical Foundations of Computer Science 2015*, Lecture Notes in Computer Science, pages 307–318, Berlin, Heidelberg, 2015. Springer.
- [KKMO16] Benjamin Lucien Kaminski, Joost-Pieter Katoen, Christoph Matheja, and Federico Olmedo. Weakest Precondition Reasoning for Expected Run-Times of Probabilistic Programs. In Peter Thiemann, editor, *Programming Languages and Systems*, Lecture Notes in Computer Science, pages 364–389, Berlin, Heidelberg, 2016. Springer.
- [KP11] Manuel Kauers and Peter Paule. *The Concrete Tetrahedron: Symbolic Sums, Recurrence Equations, Generating Functions, Asymptotic Estimates*. Texts & Monographs in Symbolic Computation. Springer-Verlag, Wien, 2011.
- [KSK76] John G. Kemeny, J. Laurie Snell, and Anthony W. Knapp. *Denumerable Markov Chains: with a chapter of Markov Random Fields by David Griffeth*. Graduate Texts in Mathematics. Springer-Verlag, New York, 2 edition, 1976.
- [MM06] Annabelle McIver and Carroll Morgan. Developing and Reasoning About Probabilistic Programs in pGCL. In Ana Cavalcanti, Augusto Sampaio, and Jim Woodcock, editors, *Refinement Techniques in Software Engineering: First Pernambuco Summer School on Software Engineering, PSSE 2004, Recife, Brazil, November 23-December 5, 2004 Revised Lectures*, Lecture Notes in Computer Science, pages 123–155. Springer, Berlin, Heidelberg, 2006.
- [MMKK17] Annabelle McIver, Carroll Morgan, Benjamin Lucien Kaminski, and Joost-Pieter Katoen. A New Proof Rule for Almost-sure Termination. *Proc. ACM Program. Lang.*, 2(POPL):33:1–33:28, December 2017.
- [NCH18] Van Chan Ngo, Quentin Carbonneaux, and Jan Hoffmann. Bounded expectations: resource analysis for probabilistic programs. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2018, pages 496–512, Philadelphia, PA, USA, June 2018. Association for Computing Machinery.

- [TOUH18] Toru Takisaka, Yuichiro Oyabu, Natsuki Urabe, and Ichiro Hasuo. Ranking and Repulsing Supermartingales for Reachability in Probabilistic Programs. *arXiv:1805.10749 [cs]*, 11138:476–493, 2018. arXiv: 1805.10749.
- [Tur37] A. M. Turing. On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, s2-42(1):230–265, January 1937. Publisher: Oxford Academic.
- [Wil91] David Williams. *Probability with Martingales*. Cambridge University Press, February 1991. Google-Books-ID: RnOJeRpk0SEC.