Quantitative Verification with Neural Networks

Alessandro Abate
University of Oxford, UK

Alec Edwards
University of Oxford, UK

Mirco Giacobbe
University of Birmingham, UK

Hashan Punchihewa
University of Oxford, UK

Diptarko Roy
University of Oxford, UK

Abstract

We present a data-driven approach to the quantitative verification of probabilistic programs and stochastic dynamical models. Our approach leverages neural networks to compute tight and sound bounds for the probability that a stochastic process hits a target condition within finite time. This problem subsumes a variety of quantitative verification questions, from the reachability and safety analysis of discrete-time stochastic dynamical models, to the study of assertion-violation and termination analysis of probabilistic programs. We rely on neural networks to represent supermartingale certificates that yield such probability bounds, which we compute using a counterexample-guided inductive synthesis loop: we train the neural certificate while tightening the probability bound over samples of the state space using stochastic optimisation, and then we formally check the certificate's validity over every possible state using satisfiability modulo theories; if we receive a counterexample, we add it to our set of samples and repeat the loop until validity is confirmed. We demonstrate on a diverse set of benchmarks that, thanks to the expressive power of neural networks, our method yields smaller or comparable probability bounds than existing symbolic methods in all cases, and that our approach succeeds on models that are entirely beyond the reach of such alternative techniques.

2012 ACM Subject Classification Theory of computation → Program verification; Theory of computation → Probabilistic computation; Computing methodologies → Machine learning; Computing methodologies → Neural networks

Keywords and phrases Data-driven Verification, Quantitative Verification, Probabilistic Programs, Stochastic Dynamical Models, Counterexample-guided Inductive Synthesis, Neural Networks

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2023.22

Related Version *FullVersion*: `https://arxiv.org/abs/2301.06136` [2]

Funding *Alessandro Abate*: HICLASS project from Innovate UK
Alec Edwards: EPSRC Centre for Doctoral Training in Autonomous Intelligent Machines and Systems (EP/S024050/1)
Hashan Punchihewa: DeepMind Computer Science Scholarship
Diptarko Roy: EPSRC Doctoral Training Partnership, and Department of Computer Science Scholarship, University of Oxford

1 Introduction

Probabilistic programs extend imperative programs with the ability to sample from probability distributions [20, 27, 31, 37], which provides an expressive language to describe randomized algorithms, cryptographic protocols, and Bayesian inference schemes. Discrete-time stochastic dynamical models, characterised by stochastic difference equations, are a natural framework

© Alessandro Abate, Alec Edwards, Mirco Giacobbe, Hashan Punchihewa, and Diptarko Roy;
licensed under Creative Commons License CC-BY 4.0
34th International Conference on Concurrency Theory (CONCUR 2023).
Editors: Guillermo A. Pérez and Jean-François Raskin; Article No. 22; pp. 22:1–22:18

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

to describe auto-regressive time series, as well as sequential decision and planning problems in unknown environments. A fundamental quantitative verification problem for probabilistic programs and stochastic dynamical models is the quantitative reachability question, which amounts to finding the probability with which the system reaches a given target condition within a finite number of steps. Reachability is at the core of a variety of other important quantitative verification questions as, by selecting appropriate target conditions on the state space, we can express the probability that a probabilistic program terminates or that it violates an assertion, as well as the probability that a stochastic dynamical model satisfies an invariant or remains within a set of safe configurations.

Quantitative reachability verification has been studied extensively using theories and algorithms built upon symbolic reasoning techniques, such as quantitative calculi [36, 38], probabilistic model checking algorithms [22, 33], discrete abstractions of stochastic dynamical models [4, 44], and the synthesis of supermartingale-like certificates [13, 15, 16, 19]. Among the latter class, a method to provide a sound upper bound for the reachability probability of a system is to synthesise a supermartingale function that maps every reachable state to a non-negative real, whose value is never smaller than 1 inside the target condition, and such that it never increases in expectation as the system evolves outside of the target. This is referred to as a *non-negative repulsing supermartingale* or *stochastic invariant indicator* in the literature [17, 19, 43], and its output over a given state provides an upper bound for the probability that the system reaches the target condition from that state. Symbolic methods for the synthesis of such certificates assume that the supermartingale function, as well as its post-expectation, which depends on the model constraints and distributions, are both in linear or polynomial form. This poses syntactic restrictions to their applicability.

Data-driven and counterexample-guided inductive synthesis (CEGIS) procedures, combined with machine learning approaches that leverage neural networks to represent certificates, have shown great promise in mitigating the aforementioned limitation [1, 3, 14, 26, 34, 35, 40]. In particular, neural-based CEGIS decouples the task of guessing a certificate from that of checking its validity, delegating the guessing task to efficient machine learning algorithms that leverage the expressive power of neural networks, while confining symbolic reasoning to the checking part of the task, which is computationally easier to solve in isolation than the entire synthesis problem. CEGIS has been applied to the synthesis of *neural supermartingales* for the almost-sure termination of probabilistic programs [3], as well as their counterpart for stochastic dynamical models with applications to qualitative queries such as almost-sure safety and stability [34, 35].

In this paper, we present theory, methods, and an extensive experimental evaluation to demonstrate the efficacy and flexibility of neural supermartingales to solve *quantitative verification* questions for probabilistic program and stochastic dynamical models, using machine learning combined with satisfiability modulo theories (SMT) technologies.

- **Theory.** We adapt the theory of non-negative repulsing supermartingales to leverage neural networks as representations of supermartingale functions for quantitative verification. Unlike previous deductive methods which need deterministic invariants and impose restrictions on the models under study, our version entirely relies on neural architectures and SMT solving to guarantee soundness, without requiring such assumptions.
- **Methods.** We present a CEGIS-based approach to train neural supermartingale functions that minimise an upper bound for the reachability probability over sample points from the state space, and check their validity over every possible state using SMT solving. We present a program-agnostic approach that relies on state samples in the training phase, and also a novel program-aware approach that embeds model information in the loss function to enhance the effectiveness of stochastic optimisation.

$v \in \text{Vars}$	(variables)
$N \in \mathbb{R}$	(numerals)
$E ::= v \mid N \mid \texttt{-}E \mid E + E \mid E - E \mid E * E \mid \ldots$	(arithmetic expressions)
$P ::= \texttt{Bernoulli(}E\texttt{)} \mid \texttt{Gaussian(}E\texttt{, }E\texttt{)} \mid \ldots$	(probability distributions)
$B ::= \texttt{true} \mid \texttt{!}B \mid B \,\texttt{\&\&}\, B \mid B \,\texttt{\|\|}\, B \mid E \texttt{==} E \mid E \texttt{<} E \mid \ldots$	(Boolean expressions)
$C ::= \texttt{skip}$	(update commands)
$\mid v \texttt{=} E$	(deterministic assignment)
$\mid v \sim P$	(probabilistic assignment)
$\mid C \,;\, C$	(sequential composition)
$\mid \texttt{if}\ B\ \texttt{then}\ C\ \texttt{else}\ C\ \texttt{fi}$	(conditional composition)

Figure 1 Grammar for update commands, Boolean and arithmetic expressions.

Experiments. We build a prototype implementation and compare the efficacy of our method with the state of the art in synthesis of linear supermartingales using symbolic reasoning [43]. We show that our program-aware approach computes tighter or comparable probability bounds than symbolic reasoning on existing benchmarks, while our program-agnostic approach matches it in over half of the instances. We additionally demonstrate that both our approaches can handle models beyond reach of purely symbolic methods.

2 Probabilistic Programs and Stochastic Dynamical Models

Probabilistic programs are computer programs whose execution is determined by random variables, and stochastic dynamical models describe discrete-time dynamical systems with probabilistic behaviour. The earlier enjoy the flexibility of imperative programming constructs and are used to describe randomised algorithms, and the latter are expressed as stochastic difference equations and are used to describe probabilistic systems that evolve over infinite time. The semantics of both can be described in terms of stochastic processes and, for this reason, verification questions for both can be solved with similar techniques.

The syntax of our modeling framework uses imperative constructs from probabilistic programs and defines executions over infinite time as dynamical systems. Specifically, we consider programs that operate over an ordered set of n real-valued variables, denoted by Vars, and update their values through the repeated execution of a command C whose grammar is described in Figure 1. Under this definition, a state of the system is an n-dimensional vector $s \in \mathbb{R}^n$ that assigns a value to each variable symbol. The update command C defines an update function $f \colon \mathbb{R}^n \times [0,1]^m \to \mathbb{R}^n$, where m is the number of syntactic probabilistic assignment statements occurring in C, which maps the current state and m random variables uniformly distributed in $[0,1]$ into the next state. Conceptually, within f, each random variable is mapped into its respective distribution by applying the appropriate inverse transformation. Altogether, our probabilistic program defines a stochastic process, whose behavior is determined by the following stochastic difference equation:

$$s_{t+1} = f(s_t, r_t), \quad r_t \sim \mathbb{U}^m, \tag{1}$$

where r_t is an m-dimensional random input sampled at time t from the uniform distribution $\mathbb{U}^m$ over the m-dimensional hypercube $[0,1]^m$. The initial state s_0 is either given as a deterministic assignment to constant values, or is non-deterministically chosen from a

set of initial conditions S_0 characterized by a Boolean expression. This setting can be seen as an assertion about the initial conditions followed by the probabilistic program `while true do` C `od`. As we show in Section 3, this allows us to characterise verification questions such as termination, non-termination, invariance and assertion-violation for while loops with general guards, as well as reachability and safety verification questions for dynamical models with general stochastic disturbances.

The semantics of our model is defined as a stochastic process induced by the Markov chain over the probability space of infinite words of random samples. This is defined by the probability space triple $(\Omega, \mathcal{F}, \mathbb{P})$ [12, 28], where

- Ω is the set of infinite sequences $([0,1]^m)^\omega$ of m-dimensional tuples of values in $[0,1]$,
- $\mathcal{F}$ is the extension of the Borel σ-algebra over the unit interval $\mathcal{B}([0,1])$ to Ω,
- $\mathbb{P}$ is the extension of the Lebesgue measure on $[0,1]$ to Ω.

Every initialisation of the system on state $s \in \mathbb{R}^n$ induces a stochastic process $\{X_t^s(\omega)\}_{t\in\mathbb{N}}$ over the state space $\mathbb{R}^n$. Let $\omega = r_0 r_1 r_2 \ldots$ be an infinite sequence of random samples in $[0,1]^m$, then the stochastic process is defined by the sequence of random variables

$$X_{t+1}^s(\omega) = f(X_t^s(\omega), r_t), \qquad X_0^s(\omega) = s. \tag{2}$$

This defines the natural filtration $\{\mathcal{F}_t\}_{t\in\mathbb{N}}$, which is the smallest filtration to which the stochastic process X_t^s is adapted. In other words [30], this can be seen as another Markov chain with state space $\mathbb{R}^n$ and transition kernel

$$T(s, S') = \text{Leb}\left(\{r \in [0,1]^m \mid f(s,r) \in S'\}\right), \tag{3}$$

where S' is a Borel measurable subset of $\mathbb{R}^n$ and Leb refers to the Lebesgue measure of a measurable subset of $[0,1]^m$. In other words, kernel T denotes the probability to transition from state s into a set of states S'. The transition kernel also defines the *post-expectation operator* $\mathbb{X}$, also known as the next-time operator [43, Definition 2.16]. $\mathbb{X}$ can be applied to an arbitrary Borel-measurable function $h\colon \mathbb{R}^n \to \mathbb{R}$ defining the *post-expectation of* h, denoted by $\mathbb{X}[h]$ and defined as the following function over states:

$$\mathbb{X}[h](s) = \int h(s') T(s, \mathrm{d}s'). \tag{4}$$

This represents the expected value of h evaluated at the next state, given the current state being s. Computing the symbolic representation of a post-expectation for probabilistic programs and stochastic dynamical models is a core problem in probabilistic verification [24, 25]. Indeed, our theoretical framework builds upon the post-expectation (cf. Eq. (13b)), and our verification procedure uses a symbolic representation of the post-expectation in our program-aware method (cf. Eqs. (17) and (19)), while our program-agnostic method approximates it statistically.

We remark that our model encompasses general probabilistic program loops as well as stochastic dynamical systems with general disturbances. For example, a probabilistic loop with guard condition B and body C in the form

`while` B `do` C `od` (5)

can be expressed as a loop with guard `true` and body `if` B `then` C `else skip fi`. This expresses the fact that, after termination, the program will stay on the terminal state indefinitely. Also, discrete-time stochastic difference equations with a nonlinear vector field g and time-invariant input disturbance with an arbitrary distribution $\mathcal{W}$ in the form

$$s_{t+1} = g(s_t, w_t), \quad w_t \sim \mathcal{W}. \tag{6}$$

comply with our model. It is sufficient to derive w_t with an appropriate inverse transformation from the uniform distribution and embed it in f. Our model is even more general, as it encompasses state-dependent distributions, whose parameters depend on the state and may depend on other distributions and thus define joint, multi-variate and hierarchically-structured distributions. Notably, our model comprises both continuous and discrete probability distributions and is able to model discrete-time stochastic hybrid systems.

3 Quantitative Reachability Verification of Probabilistic Models

Quantitative verification treats the question of providing a quantity for which a system satisfies a property as opposed to providing a definite positive or negative answer. In fact, it is sometimes too conservative or inappropriate to demand a definite outcome to a formal verification question. For instance, a system whose behaviour is probabilistic may violate a specification on rare corner cases and yet satisfy it with a probability that is deemed acceptable for the application domain. We address the quantitative verification of probabilistic systems, which is the problem of computing the probability for which a system satisfies a specification [7, 8, 32, 45].

We consider the *quantitative reachability verification* question, which as we show below, is at the core of a variety of quantitative verification questions for probabilistic programs and stochastic dynamical models. Henceforth, we use $\mathbf{1}_S$ to denote the indicator function of set S, i.e., $\mathbf{1}_S(s) = 1$ if $s \in S$ and $\mathbf{1}_S(s) = 0$ if $s \notin S$; we also use $\lambda x.M$ to denote the anonymous function that takes an argument x and evaluates the expression M to produce its result. We now characterise the probability that a stochastic process reaches a Borel measurable target set A in exactly time t, in at most time t, and in any finite time.

▶ **Lemma 1.** *Let the event that a stochastic process $\{X_t^s(\omega)\}_{t\in\mathbb{N}}$ over state space $\mathbb{R}^n$ initialised in state $s \in \mathbb{R}^n$ reaches a target set $A \in \mathcal{B}(\mathbb{R}^n)$ in exactly time $t \in \mathbb{N}$ be*

$$\mathrm{Reach}_t^s(A) = \left\{\omega \in \Omega \mid X_0^s(\omega) \notin A, \dots, X_{t-1}^s(\omega) \notin A, X_t^s(\omega) \in A\right\}, \tag{7}$$

with $\mathrm{Reach}_0^s(A) = \Omega$ *if* $s \in A$*, and* $\mathrm{Reach}_0^s(A) = \emptyset$ *if* $s \notin A$*. Then,* $\mathrm{Reach}_t^s(A)$ *is measurable and its probability measure can be expressed as follows:*

$$\mathbb{P}[\mathrm{Reach}_{t+1}^s(A)] = \mathbf{1}_{\mathbb{R}^n\setminus A}(s) \cdot \mathbb{X}[\lambda s'.\, \mathbb{P}[\mathrm{Reach}_t^{s'}(A)]](s), \tag{8a}$$

$$\mathbb{P}[\mathrm{Reach}_0^s(A)] = \mathbf{1}_A(s). \tag{8b}$$

▶ **Lemma 2.** *Let the event that a stochastic process $\{X_t^s(\omega)\}_{t\in\mathbb{N}}$ over state space $\mathbb{R}^n$ initialised in state $s \in \mathbb{R}^n$ reaches a target set $A \in \mathcal{B}(\mathbb{R}^n)$ in at most time $t \in \mathbb{N}$ be*

$$\mathrm{Reach}_{\le t}^s(A) = \cup\{\ \mathrm{Reach}_i^s(A) \colon 0 \le i \le t\ \}, \tag{9}$$

Then, $\mathrm{Reach}_{\le t}^s(A)$ *is measurable and its probability measure can be expressed as follows:*

$$\mathbb{P}[\mathrm{Reach}_{\le t+1}^s(A)] = \mathbf{1}_A(s) + \mathbf{1}_{\mathbb{R}^n\setminus A}(s) \cdot \mathbb{X}[\lambda s'.\, \mathbb{P}[\mathrm{Reach}_{\le t}^{s'}(A)]](s), \tag{10a}$$

$$\mathbb{P}[\mathrm{Reach}_{\le 0}^s(A)] = \mathbf{1}_A(s). \tag{10b}$$

▶ **Lemma 3.** *Let the event that a stochastic process $\{X_t^s(\omega)\}_{t\in\mathbb{N}}$ over state space $\mathbb{R}^n$ initialised in state $s \in \mathbb{R}^n$ reaches a target set $A \in \mathcal{B}(\mathbb{R}^n)$ in finite time be*

$$\mathrm{Reach}_{\mathrm{fin}}^s(A) = \cup\{\mathrm{Reach}_t^s(A) \colon t \in \mathbb{N}\} = \cup\left\{\mathrm{Reach}_{\le t}^s(A) \colon t \in \mathbb{N}\right\}. \tag{11}$$

Then, $\mathrm{Reach}^{s}_{\mathrm{fin}}(A)$ *is measurable and its probability measure (which we refer to as the reachability probability of the target set) can be expressed as follows:*

$$\mathbb{P}[\mathrm{Reach}^{s}_{\mathrm{fin}}(A)] = \lim_{t\to\infty} \mathbb{P}[\mathrm{Reach}^{s}_{\leq t}(A)]. \tag{12}$$

The lemmata above follow from measure-theoretic results in probabilistic verification (proofs are provided in the extended version [2]), and underpin the formal characterisation of quantitative probabilistic reachability in the next section. Our method leverages neural networks to compute an upper bound for the reachability probability with respect to a given target set (cf. Section 4). Notice that an appropriate choice of target set allows us to express a variety of other quantitative verification questions, for which our method can provide an upper or lower bound. Below, by providing a suitable choice for the target set, we show how important quantitative verification questions for probabilistic programs and stochastic dynamical models can be characterised as instances of quantitative reachability.

Termination Analysis. Let $G \in \mathcal{B}(\mathbb{R}^n)$ be the guard set of a probabilistic while loop as in Eq. (5), i.e., the set of states for which that guard condition evaluates to true. The event that the loop terminates from initial state s_0 is $\mathrm{Reach}^{s_0}_{\mathrm{fin}}(\mathbb{R}^n \setminus G)$. Our method computes an upper bound for the probability that the loop terminates or, dually, it computes a lower bound for the probability of non-termination. Notably, when this lower bound is greater than 0, then almost-sure termination is refuted.

Assertion-violation Analysis. Let $G \in \mathcal{B}(\mathbb{R}^n)$ be the guard set of a probabilistic while loop and $A \in \mathcal{B}(\mathbb{R}^n)$ be the satisfying set of an assertion placed at the beginning of the loop body. Given initial state s_0, the event that the assertion is eventually violated is $\mathit{Reach}^{s_0}_{\mathrm{fin}}(G \setminus A)$. Our method computes an upper bound for the probability of assertion violation or, dually, a lower bound for its satisfaction. Note that assertions in other positions of the body can be modelled similarly by using additional Boolean variables.

Safety Verification. Let $B \in \mathcal{B}(\mathbb{R}^n)$ be a set of undesirable states in a stochastic dynamical model. The event that the system is safe when initialised in s_0, i.e., it never reaches an undesirable state, is given by $\Omega \setminus \mathit{Reach}^{s_0}_{\mathrm{fin}}(B)$. Our method computes a lower bound for the probability that the system is safe, which is $1 - \mathbb{P}[\mathit{Reach}^{s_0}_{\mathrm{fin}}(B)]$.

Invariant Verification. Let $I \in \mathcal{B}(\mathbb{R}^n)$ be a candidate invariant set. The event that I is invariant when the system is initialised in s_0 is $\Omega \setminus \mathit{Reach}^{s_0}_{\mathrm{fin}}(\mathbb{R}^n \setminus I)$. Our method computes a lower bound for the probability that set I is invariant, which is $1 - \mathbb{P}[\mathit{Reach}^{s_0}_{\mathrm{fin}}(\mathbb{R}^n \setminus I)]$. Note that if $s_0 \notin I$, this definition yields a trivial lower bound of zero for the probability of invariance.

4 Neural Supermartingales for Quantitative Verification

Supermartingale certificates provide a flexible and powerful theoretical framework for the formal verification of probabilistic models with infinite state spaces. Not only have supermartingales been applied to qualitative questions, such as almost-sure termination analysis of probabilistic programs and almost-sure stability analysis of stochastic dynamical models, but also to quantitative reachability verification, as in this work. Specifically, this is enabled by the theory of non-negative repulsing supermartingales and of stochastic invariants [17, 19, 43].

Our method builds upon the theory of non-negative repulsing supermartingales and stochastic invariant indicators and adapts it to take advantage of the expressive power of neural networks and the flexibility of machine learning and counterexample-guided inductive synthesis algorithms. Our method hinges on the following theorem, whose proof we provide in the extended version [2].

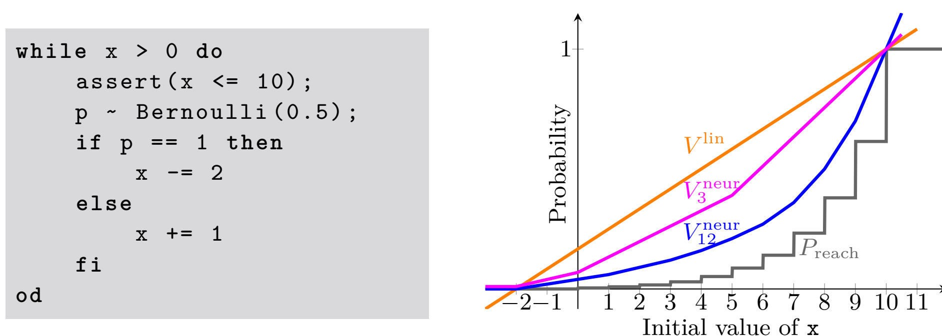


Figure 2 Comparison between linear and neural supermartingale functions in tightness of bounds for the assertion violation probability of the program repulse, shown on the left. On the right, the function V^{lin} indicates the tightest linear supermartingale (under the restriction that x is greater than -2). The functions V_3^{neur} and V_{12}^{neur} indicate single-layer neural supermartingales with 3 and 12 neurons respectively. The piecewise constant function P_{reach} is the true probability of assertion violation, and indicates the ideal lower bound for the value of any supermartingale function.

▶ **Theorem 4.** *Let $\mathbb{X}$ be the post-expectation operator of a stochastic process over state space $\mathbb{R}^n$ and let $A \in \mathcal{B}(\mathbb{R}^n)$ be a target set. Let $V\colon \mathbb{R}^n \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ be a non-negative function that satisfies the following two conditions:*

$$\text{(indicating condition)} \qquad \forall s \in A\colon V(s) \geq 1, \tag{13a}$$

$$\text{(non-increasing condition)} \qquad \forall s \notin A\colon \mathbb{X}[V](s) \leq V(s), \tag{13b}$$

Then, for every state $s \in \mathbb{R}^n$, it holds that $V(s) \geq \mathbb{P}[\mathrm{Reach}^s_{\mathrm{fin}}(A)]$.

As we show in detail in Section 5, we compute a supermartingale that satisfies the two criteria (13a) and (13b), while also minimising its output over the initial state s_0. When the initial state is chosen nondeterministically from a set S_0, we instead minimise the output over all states in the set. As a consequence, the maximum of V over S_0 is a sound upper bound for the reachability probability.

In this work, we template V as a neural network with n input neurons, l hidden layers with respectively $h_1, \ldots h_l$ neurons in each hidden layer, and 1 output neuron. We guarantee a-priori that the function's output will be non-negative over the entire domain using the following architecture:

$$V(x) = \mathrm{sum}\left(\sigma_l \circ \cdots \circ \sigma_1(x)\right), \quad \sigma_i(z) = (\mathrm{ReLU}(\mathbf{w}_{i,1}^T z + \mathbf{b}_{i,1}), \ldots, \mathrm{ReLU}(\mathbf{w}_{i,h_i}^T z + \mathbf{b}_{i,h_i})) \tag{14}$$

where $\mathrm{sum}(z_{l,1}, \ldots, z_{l,h_l}) = \sum_{k=1}^{h_l} z_{l,k}$ and $\mathbf{w}_{i,j}^T \in \mathbb{Q}^{h_{i-1}}$ (defining $h_0 = n$, the number of input neurons) and $\mathbf{b}_{i,j} \in \mathbb{Q}$ are respectively the weight vector and the bias parameter for the inputs to neuron j at layer i. Then, our method trains the neural network to satisfy the two criteria (13a) and (13b), while also minimising its output on the initial condition.

Using neural networks as templates of supermartingale functions introduces non-trivial advantages with respect to symbolic methods for the synthesis of supermartingales. Firstly, neural supermartingales are able to better approximate the true probability of reachability and thus attain tighter upper bounds on it. Secondly, symbolic methods require deterministic invariants that overapproximate the set of reachable states, and suitably restricts the domain of the template. Figure 2 illustrates using an example the advantages of neural certificates with respect to linear supermartingales synthesised using Farkas' lemma. This example shows that increasing the number of neurons provides greater flexibility and allows the certificate

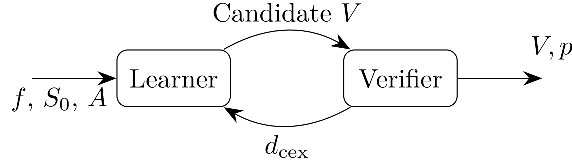


Figure 3 Overview of the counterexample-guided inductive synthesis procedure used to synthesise neural supermartingales for quantitative reachability verification. Inputs to the procedure are a probabilistic program f, a set of initial states S_0, and a target set A. The procedure outputs a valid neural supermartingale V and a probability bound p.

to more tightly approximate the true probability. Moreover, this example shows that linear supermartingales require their domain to be restricted with an appropriate deterministic invariant. Notably, symbolic methods for the synthesis of polynomial supermartingales based on Putinar's Positivstellensatz also require compact deterministic invariants to be provided [17]. By contrast, neural supermartingales (whose output is always non-negative) achieve the same result while relaxing the requirement of providing an invariant beforehand.

5 Data-driven Synthesis of Neural Supermartingales

Our approach to synthesising neural supermartingales for quantitative verification utilises a counterexample-guided inductive synthesis (CEGIS) procedure [41, 42] (cf. Figure 3). This procedure consists of two components, a learner and a verifier, that work in opposition to each other. On the one hand, the learner seeks to synthesise a candidate supermartingale that meets the desired specification (cf. Eq. (13)) over a finite set of samples from the state space, while simultaneously optimising the tightness of the probability bound. On the other hand, the verifier seeks to disprove the validity of this candidate by searching for counterexamples, i.e., instances where the desired specification is invalidated, over the entire state space. If the verifier shows that no such counterexample exists, then the desired specification is met by the supermartingale and the procedure provides a sound probability bound for the reachability probability of interest, together with a neural supermartingale to certify it.

5.1 Training of Neural Supermartingales From Samples

Our neural supermartingale for quantitative reachability verification consists of a neural network with ReLU activation functions, with an arbitrary number of hidden layers (cf. Section 4). We train this neural network using gradient descent over a finite set $D = \{d^{(1)}, \ldots, d^{(m)}\} \subseteq \mathbb{R}^n$ of states $d^{(i)}$ sampled over the state space $\mathbb{R}^n$. Initially, we sample uniformly within a bounded hyper-rectangle of $\mathbb{R}^n$, a technique that scales effectively to high dimensional state spaces and efficiently populates the initial dataset of state samples. Then, we construct a loss function that guides gradient descent to optimise the parameters (weight and biases) of the neural network V to satisfy the specification set out in Eq. (13) while also minimising the probability bound. We define a loss function $\mathcal{L}(D)$ that consists of three terms:

$$\mathcal{L}(D) = \beta_1 \mathcal{L}_{\text{ind}}(D) + \beta_2 \mathcal{L}_{\text{non-inc}}(D) + \beta_3 \mathcal{L}_{\text{min}}(D). \tag{15}$$

Components $\mathcal{L}_{\text{ind}}$ and $\mathcal{L}_{\text{non-inc}}$ are responsible for encouraging satisfaction of the conditions in Eq. (13), while the component $\mathcal{L}_{\text{min}}$ is responsible for tightening the probability bound. The parameters of this optimisation problem are the parameters of the neural network, which are initialised randomly. The dataset consists of the state samples, initially sampled randomly,

and generated from counterexamples in subsequent CEGIS iterations (cf. Section 5.2). The coefficients β_1, β_2 and β_3 denote scale factors for each term, which we choose according to the priority that we want to assign to each condition (cf. Section 6).

First, consider the condition in Eq. (13a), which we refer to as the *indicating condition*. For this, we use the following loss function:

$$\text{(indicating loss)} \qquad \mathcal{L}_{\text{ind}}(D) = \mathbb{E}_{d \in D \cap A}[\text{ReLU}(1 - V(d))]. \tag{16}$$

This adds a penalty for states $d \in D$ lying inside the target condition A, at which V fails to satisfy the indicating condition, whilst ignoring any states where V satisfies it. We average this per-state penalty across all states in $D \cap A$.

We next consider the *non-increasing condition* in Eq. (13b), for which we use the following loss term:

$$\text{(non-increasing loss)} \qquad \mathcal{L}_{\text{non-inc}}(D) = \mathbb{E}_{d \in D \setminus A}[\text{ReLU}(\mathbb{X}[V](d) - V(d))]. \tag{17}$$

This penalises states $d \in D$ lying outside of the target set A, at which V fails to satisfy the non-increasing condition. Notably, this component is defined in terms of the post-expectation $\mathbb{X}[V]$ of our supermartingale. To embed this expression in our loss function we consider two alternative approaches, which we call *program-aware* and *program-agnostic*.

Program-aware Approach. The program-aware approach uses the source code of the program to generate a symbolic expression for the post-expectation of V. For this purpose, we exploit the symbolic inference algorithm introduced by the tool PSI [24, 25], along with a symbolic representation of V. We construct a probabilistic program which represents the evaluation of V on the state resulting after the execution of the update function f. The expected value of this program is precisely $\mathbb{X}[V]$. This results in a symbolic expression that is a function of the program state and parameters of V. In the non-increasing loss, states are instantiated to elements of $D \setminus A$, while the parameters are left as free-variables that the gradient descent engine differentiates with respect to.

Program-agnostic Approach. The program-agnostic approach provides an alternative formulation of the non-increasing loss term that does not require symbolic reasoning. Instead of leveraging the program's source code, it only requires the ability to execute it. For this, we utilise a Monte Carlo scheme to estimate the post-expectation. For each state d in our dataset D, to obtain an estimate of $\mathbb{X}[V](d)$ we sample a number m' of successor states $D' = \{d'^{(k)} : 1 \leq k \leq m'\}$. Each successor state $d'^{(k)}$ is sampled by executing the program's update function f (cf. Eq. (1)) at state d. Then $\mathbb{X}[V](d)$ is estimated as $\mathbb{X}[V](d) \approx \mathbb{E}_{d' \in D'}[V(d')]$ which is the average of V over D'. Even though this is an approximation, we emphasise that this does not affect the soundness of our scheme, which is ensured by the verifier.

Finally, we introduce a tightness criterion that minimises the probability upper bound:

$$\text{(minimisation loss)} \qquad \mathcal{L}_{\min}(D) = \mathbb{E}_{d \in D \cap S_0}[V(d)]. \tag{18}$$

This term encourages V to take smaller values over the set of initial states S_0. Recall that the smaller the probability upper bound, the closer it is to the true value.

The loss function is provided to the gradient descent optimiser, whose performance benefits from a smooth objective. For this reason, to improve the performance of our learner we replace every ReLU with a smooth approximation, Softplus, which takes the form $\text{Softplus}(s) = \log(1 + \exp(s))$. Additionally, we improve the approximation of Softplus to ReLU at small values over the interval $[0, 1]$ by re-scaling V. This means that we modify the

bounding condition to require that $V(x) \geq \alpha$ over the target set A, for some large $\alpha > 1$. In other words, we modify the indicating loss to $\mathcal{L}_{\text{ind}}(V) = \mathbb{E}_{d \in D \cap A}[\text{ReLU}(\alpha - V(d))]$. We remark that while Softplus is used as the activation function in the learning stage, for the verification stage we instead employ ReLU activation functions, ensuring soundness of the generated neural supermartingale. We remark that this has no effect on the soundness of our approach, which is ultimately guaranteed by the verifier.

5.2 Verification of Neural Supermartingales Using SMT Solvers

The purpose of the verification stage is to check that the neural supermartingale meets the requirements of Eq. (13) over the entire state space $\mathbb{R}^n$, and if that is determined to be the case to furthermore obtain a sound upper bound on the reachability probability. We achieve this by constructing a suitable formula in first-order logic, Eq. (19), and use SMT solving to decide its validity, or equivalently, to decide the satisfiability of its negation, Eq. (20).

The conditions pertaining to the validity of the neural supermartingale, given in (13a) and (13b), are encoded by the formulas φ_{ind} and $\varphi_{\text{non-inc}}$. We also note that for a constant $p \in [0,1]$ to be a sound upper bound on the reachability probability, it is sufficient to require that p is an upper bound on the neural supermartingale's value over the set of initial states S_0 (cf. Theorem 4), which is expressed by the formula φ_{bound}. A suitable choice for the bound p is determined by a binary search over the interval $[0,1]$.

$$\forall s \in \mathbb{R}^n : \underbrace{(s \in A \rightarrow V(s) \geq 1)}_{\varphi_{\text{ind}}} \wedge \underbrace{(s \notin A \rightarrow \mathbb{X}[V](s) \leq V(s))}_{\varphi_{\text{non-inc}}} \wedge \underbrace{(s \in S_0 \rightarrow V(s) < p)}_{\varphi_{\text{bound}}}. \quad (19)$$

Here, $V(s)$ is a symbolic encoding of the candidate neural supermartingale proposed by the learner (Section 5.1), and S_0 and A are defined by Boolean predicates over program variables, all of which (in our setting of networks composed from ReLU activations) are expressible using expressions and constraints in non-linear real arithmetic. For this reason, we use Z3 as our SMT solver [21].

The SMT solver is provided with the negation of Eq. (19), namely

$$\exists s \in \mathbb{R}^n : \underbrace{(s \in A \wedge V(s) < 1)}_{\neg\varphi_{\text{ind}}} \vee \underbrace{(s \notin A \wedge \mathbb{X}[V](s) > V(s))}_{\neg\varphi_{\text{non-inc}}} \vee \underbrace{(s \in S_0 \wedge V(s) \geq p)}_{\neg\varphi_{\text{bound}}}, \quad (20)$$

and decides its satisfiability, seeking an assignment d_{cex} of s that is a counterexample to the neural supermartingale's validity, for which any of $\neg\varphi_{\text{ind}}$, $\neg\varphi_{\text{non-inc}}$ and $\neg\varphi_{\text{bound}}$ are satisfied. If no counterexample is found, this certifies the validity of the neural supermartingale. Alternatively, if a counterexample d_{cex} is found, it is added to the data set D, for the synthesis to incrementally resume.

6 Experimental Evaluation

The previous section develops a method for synthesising neural supermartingales. This section presents an empirical evaluation of the method, by testing it against a series of benchmarks. Each benchmark is tested ten times. A test is successful if a valid supermartingale is synthesised, and the proportion of successful tests is recorded. Further, the average bound from the valid supermartingales is also recorded, along with the average time taken by the learning and verification steps, respectively. This procedure is applied separately for program-aware and program-agnostic synthesis. We use values $\beta_1 = 10, \beta_2 = 10^5, \beta_3 = 1$ in Eq. (15) to define the priority ordering of the three terms in the loss function, which we find beneficial across our set of benchmarks. To compare our method against existing work, we

Table 1 Results comparing neural supermartingales with Farkas Lemma for different benchmarks. Here, p is the average probability bound generated by the certificate; success ratio is the number of successful experiments, out of 10 repeats, generated by CEGIS with neural supermartingale; '-' means no result was obtained. We also denote the architecture of the network: (h_1, h_2) denotes a network with 2 hidden layers consisting of h_1 and h_2 neurons respectively.

Benchmark	**Farkas' Lemma**	**Quantitative Neural Certificates**				**Network Arch.**
		Program-Agnostic		Program-Aware		
		p	Success Ratio	p	Success Ratio	
`persist_2d`	-	≤ 0.1026	0.9	≤ 0.1175	0.9	(3, 1)
`faulty_marbles`	-	≤ 0.0739	0.9	≤ 0.0649	0.8	3
`faulty_unreliable`	-	≤ 0.0553	0.9	≤ 0.0536	1.0	3
`faulty_regions`	-	≤ 0.0473	0.9	≤ 0.0411	0.9	(3, 1)
`cliff_crossing`	≤ 0.4546	≤ 0.0553	0.9	≤ 0.0591	0.8	4
`repulse100`	≤ 0.0991	≤ 0.0288	1.0	≤ 0.0268	1.0	3
`repulse100_uniform`	≤ 0.0991	≤ 0.0344	1.0	-	-	2
`repulse100_2d`	≤ 0.0991	≤ 0.0568	1.0	≤ 0.0541	1.0	3
`faulty_varying`	≤ 0.1819	≤ 0.0864	1.0	≤ 0.0865	1.0	2
`faulty_concave`	≤ 0.1819	≤ 0.1399	1.0	≤ 0.1356	0.9	(3, 1)
`fixed_loop`	≤ 0.0091	≤ 0.0095	1.0	≤ 0.0094	1.0	1
`faulty_loop`	≤ 0.0181	≤ 0.0195	1.0	≤ 0.0184	1.0	1
`faulty_uniform`	≤ 0.0181	≤ 0.0233	1.0	≤ 0.0221	1.0	1
`faulty_rare`	≤ 0.0019	≤ 0.0022	1.0	≤ 0.0022	1.0	1
`faulty_easy1`	≤ 0.0801	≤ 0.1007	1.0	≤ 0.0865	1.0	1
`faulty_ndecr`	≤ 0.0561	≤ 0.0723	1.0	≤ 0.0630	1.0	1
`faulty_walk`	≤ 0.0121	≤ 0.0173	1.0	≤ 0.0166	1.0	1

perform template-based synthesis of linear supermartingales using Farkas' Lemma. This requires deterministic invariants to overapproximate the reachable set of states, which may either be generated by abstract interpretation, or provided manually [43]. In our experiments, we provide a suitable invariant manually based on the guard of the loop, in some cases strengthening them with additional constraints by an educated guess.

It should be noted that our method is inherently stochastic. One reason is the random initialisation of the neural template's parameters in the learning phase. In program-agnostic synthesis, an additional source of randomness is the sampling of successor states. So that the results accurately reflect the performance of our method, the random seed for these sources of randomness is selected differently for each test. An additional source of non-determinism arises from the SMT solver Z3 as it generates counterexamples: this cannot be controlled externally. Benchmarks are run on a machine with an Nvidia A40 GPU, and involve the assertion-violation analysis of programs created using the following two patterns:

Unreliable Hardware. These are programs that execute on unreliable hardware. The goal is to upper bound the probability that the program fails to terminate due to a hardware fault. A simple example is `faulty_loop`, whose source code is presented in the extended version [2], and which consists of a loop which may violate an assertion with small probability, modelling a hardware fault.

Robot Motion. These programs model an agent (e.g., a robot) that moves within a physical environment. In these benchmarks, the uncertainty in control and sensing is modelled probabilistically. The environment contains a target region and a hazardous region. The goal is to upper bound the probability that the robot enters the hazardous region. We provide the program `repulse100` as an example, which is variant of `repulse` (Figure 2)

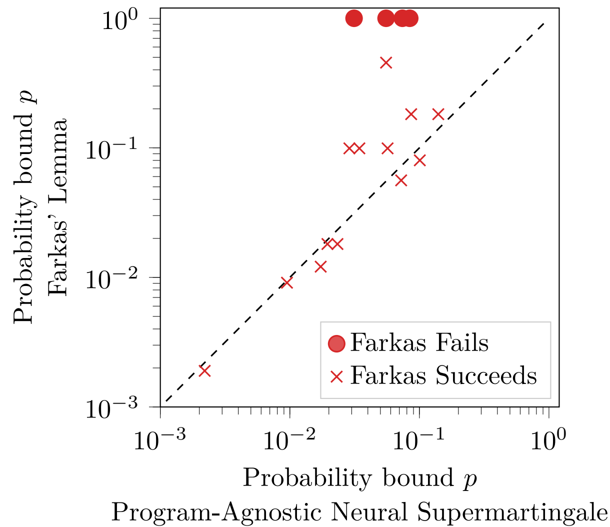


Figure 4 Probability bounds generated using program-agnostic neural supermartingales and using Farkas' Lemma. The reference line $y = x$ allows one to see which approach outperforms the other and by how much: above the line means that neural supermartingales outperform Farkas' Lemma, below the line the opposite. Neural supermartingales can significantly outperform linear templates when a better bound exists, but otherwise achieve similar results. Our approach with program-aware neural supermartingales provides even better outcomes, compared with Farkas' Lemma.

containing a modified assertion and initial state. The program models the motion of a robot in a one-dimensional environment, starting at $\texttt{x} = 10$. The target region is where $\texttt{x} < 0$, and hazardous region is where $\texttt{x} > 100$. As with `repulse`, in each iteration there is an equal probability of `x` being decremented by 2, and x being incremented by 1.

Several of the programs are based on benchmarks used in prior work focusing on other types of supermartingales [6, 13]. Additionally, there are several benchmarks that are entirely new.

The results are reported in Table 1. The table is divided into three sections. The first section shows the benchmarks where Farkas' Lemma cannot be applied, and where only our method is capable of producing a bound. The second section shows examples where both methods are able to produce a bound, but our method produces a notably better bound. The third section shows benchmarks where both methods produce comparable bounds.

Dashes in the table indicate experiments where a valid supermartingale could not be obtained. In the case of Farkas' Lemma, there are several cases where there is no linear supermartingale for the benchmark. By contrast, program-agnostic and program-aware synthesis could be applied to almost all benchmarks. The one exception is `repulse100_uniform` where only program-aware verification was unsuccessful: this is due to indicator functions in the post-expectation, which are not smooth and posed a problem for the optimiser. This benchmark underscores the value of program-agnostic synthesis, since it does not require embedding the explicit post-expectation in the loss function.

The first section of the benchmarks in Table 1 demonstrates that our method produces useful results on programs that are out-of-scope for existing techniques. Furthermore, the success ratio of our method is high on all the benchmarks, which indicates its robustness. For the second and third sections (which consist of benchmarks to which Farkas' Lemma is applicable), the success ratio of our method is broadly maximal, which is to be expected, since these programs can also be solved by Farkas' Lemma.

Table 2 Results showing the time taken in seconds to synthesise supermartingales by our method and Farkas' Lemma. For our method, we show the time taken during learning and verification.

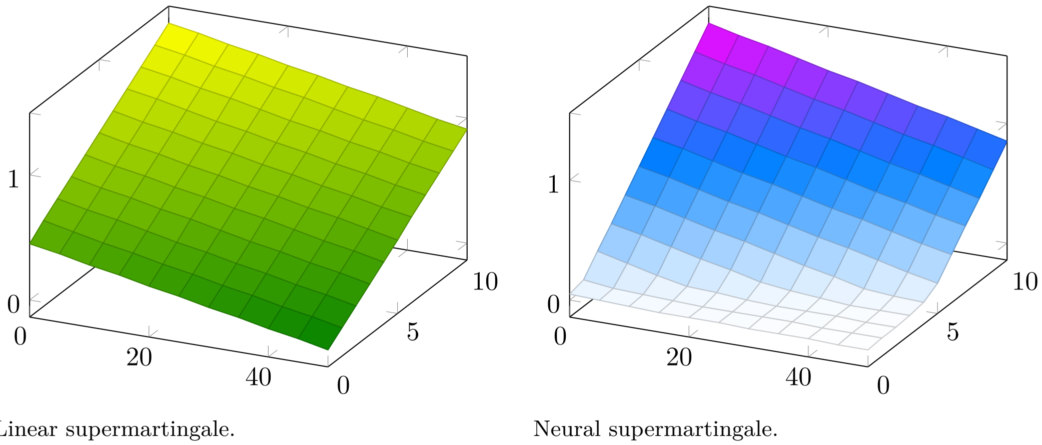
Benchmark	Farkas' Lemma	Quantitative Neural Certificates			
		Program-Agnostic		Program-Aware	
		Learn Time	Verify Time	Learn Time	Verify Time
persist_2d	-	169.14	85.31	44.96	74.90
faulty_marbles	-	114.24	29.23	15.86	28.68
faulty_unreliable	-	123.85	45.48	18.34	33.97
faulty_regions	-	17.92	35.85	17.55	32.38
cliff_crossing	0.11	134.61	19.02	21.27	29.07
repulse100	0.19	16.65	5.00	6.49	3.74
repulse100_uniform	0.19	21.28	14.18	-	-
repulse100_2d	0.12	122.92	64.54	15.75	47.70
faulty_varying	0.36	21.74	5.06	4.71	3.28
faulty_concave	0.39	49.12	13.37	13.49	7.82
fixed_loop	0.15	14.16	3.14	3.34	2.43
faulty_loop	0.16	25.52	3.81	3.73	2.66
faulty_uniform	0.34	20.20	1.91	6.75	1.33
faulty_rare	0.27	25.52	4.27	3.71	2.96
faulty_easy1	0.31	104.20	12.78	4.95	7.51
faulty_ndecr	0.33	104.89	9.06	5.37	4.66
faulty_walk	0.32	15.08	4.00	6.97	3.33

In the second section of Table 1, we find more complex benchmarks where our method was able to significantly improve the bound from Farkas' Lemma. The smallest improvement was about 0.04, and the largest improvement was over 0.39. The intuition here is that neural templates allow more sophisticated supermartingales to be learnt, that can approximate how the reachability probability varies across the state space better than linear templates, and thereby yield tighter probability bounds.

The third section of Table 1 consists of relatively simple benchmarks, where our method produces results that are marginally less tight in comparison to Farkas' Lemma. This is not surprising since our method uses neural networks consisting of a single neuron for these examples, owing to their simplicity. The expressive power of these networks is therefore similar to linear templates.

In summary, the results show that our method does significantly better on more complex examples, and marginally worse on very simple examples. This is highlighted in Figure 4. Each point represents a benchmark. The position on the x-axis shows the probability bound obtained by our program-agnostic method, and the y-axis shows the probability bound obtained by Farkas' Lemma. Points above the line indicate benchmarks where neural supermartingales outperform Farkas' Lemma, and vice versa. The scale is logarithmic to emphasise order-of-magnitude differences.

Notice that in Table 1 the program-aware algorithm usually yields better bounds than the program-agnostic algorithm, but the improvement is mostly marginal. This is in fact a strength of our method: our data-driven approach performs almost as well as one dependent on symbolic representations, which is promising in light of questions of scalability to more complex programs. We also include a breakdown of computation time (Table 2) which allows distinguishing between learning and verification overheads. Notably, Farkas' Lemma



Linear supermartingale.

Neural supermartingale.

Figure 5 Supermartingale functions for the `cliff_crossing` benchmark as generated using Farkas' Lemma (on the left) and using neural supermartingales (on the right). The right hand figure illustrates the tighter bounds obtainable through the use of neural templates.

is significantly faster than our method, given that it relies on solving a convex optimisation problem via linear programming, whereas the synthesis of neural supermartingales is a non-convex optimisation problem that is addressed using gradient descent.

Having presented the experimental results, we shall further comment on some specific benchmarks. The `repulse100` program in Table 1 is a variation of `repulse` presented in Figure 2, but with the assertion changed to `assert(x <= 100)`. While this is a small program, our method is still able to produce a significantly better result than Farkas' lemma, using a neural supermartingale with a single hidden layer consisting of three ReLU components that are summed together, which allows a convex piecewise linear function to be learnt. The `cliff_crossing` program is a further benchmark for which our method is capable of producing a significantly better bound. This is a 2 dimensional benchmark, for which we use a neural supermartingale that consists of two input neurons and four ReLU components, leading to a clear improvement compared to the linear supermartingale in the tightness of the probability bounds generated, as illustrated by Figure 5. Both `repulse100` and `cliff_crossing` are benchmarks that use neural supermartingales with a single hidden layer. An example that uses two hidden layers is `faulty_concave`, in which there are two distinct regions of the state space, one of which has a significantly higher reachability probability than the other. We find that neither a linear template nor a single-layer neural supermartingale is able to exploit this conditional behaviour, each of which yield an overly conservative certificate, but that a neural supermartingale with two hidden layers is able to more tightly approximate the reachability probability in each of the two regions. Further discussion and the source code of these case studies are presented in the extended version [2].

7 Related Work

The formal verification of probabilistic programs using supermartingales is a well-studied topic. Early approaches to introduce this technique applied them to almost-sure termination analysis of probabilistic programs [13], which allowed several extensions to polynomial programs, programs with non-determinism, lexicographic and modular termination arguments, and persistence properties [5, 15, 16, 19, 23, 29]. All these methods relied on symbolic reasoning algorithms for synthesising supermartingales, that leveraged theories based on Farkas' lemma

for the synthesis of linear certificates, and Putinar's Positivestellensatz and sum-of-square methods for the synthesis of polynomial certificates. While these methods are the state-of-the-art for many existing problem instances in literature, to achieve the strong guarantees that they provide (such as completeness for the specific class of programs they target), they must necessarily introduce restrictions on the class of programs to which they are applicable, and the form of certificates that they derive. Moreover, symbolic methods need externally provided invariants that are stronger than $\mathbb{R}^n$ to enforce non-negativity in the case of linear certificates, as we illustrate in Figure 2. Also, symbolic methods for the synthesis of polynomial certificates require compact deterministic invariants to operate.

The use of neural networks to represent certificates has allowed many of these restrictions to be lifted. In the context of the analysis of probabilistic programs, neural networks were first applied to certify positive almost-sure termination [3]. This approach lent itself to a wider range of formal verification questions for stochastic dynamical models, from stability and safety analysis to controller synthesis [18, 34, 35]. These data-driven inductive synthesis techniques for supermartingales have also been extended to machine learning techniques other then deep learning, such as piecewise linear regression and decision tree learning [10, 11].

In this paper, we further extend data-driven synthesis of neural supermartingale certificates to quantitative verification questions. The correctness of our approach builds upon the theory of non-negative repulsing supermartingales [43], as formulated in Theorem 4. Our experiments have demonstrated that neural certificates attain comparable results on programs that are amenable to symbolic analysis (such as those in the third section of Table 1), while surpassing symbolic methods on more complex programs that are either out-of-scope or yield overly conservative bounds when existing techniques are applied (such as those in the first and second section of Table 1).

8 Conclusion

We have presented a data-driven framework for the quantitative verification of probabilistic models that leverage neural networks to represent supermartingale certificates. Our experiments have shown that neural certificates are applicable to a wider range of probabilistic models than was previously possible using purely symbolic techniques. We also illustrate that on existing models our method yields certificates of better or comparable quality than those produced by symbolic techniques for the synthesis of linear supermartingales. This builds upon the ability of neural networks to approximate non-linear functions, while satisfying the constraints imposed by Theorem 4 without the need for supporting deterministic invariants to be provided externally. Our method applies to quantitative termination and assertion-violation analysis for probabilistic programs, as well as safety and invariant verification for stochastic dynamic models. We imagine extensions to further quantitative verification questions, such as temporal properties beyond reachability [9], and bounding expected accrued costs [39, 46, 47].

References

1 Alessandro Abate, Daniele Ahmed, Mirco Giacobbe, and Andrea Peruffo. Formal synthesis of Lyapunov neural networks. *IEEE Control. Syst. Lett.*, 5(3):773–778, 2021.

2 Alessandro Abate, Alec Edwards, Mirco Giacobbe, Hashan Punchihewa, and Diptarko Roy. Quantitative verification with neural networks, 2023. `arXiv:2301.06136`.

3 Alessandro Abate, Mirco Giacobbe, and Diptarko Roy. Learning probabilistic termination proofs. In *CAV (2)*, volume 12760 of *Lecture Notes in Computer Science*, pages 3–26. Springer, 2021.

4 Alessandro Abate, Joost-Pieter Katoen, and Alexandru Mereacre. Quantitative automata model checking of autonomous stochastic hybrid systems. In *HSCC*, pages 83–92. ACM, 2011.

5 Sheshansh Agrawal, Krishnendu Chatterjee, and Petr Novotný. Lexicographic ranking supermartingales: an efficient approach to termination of probabilistic programs. *Proc. ACM Program. Lang.*, 2(POPL):34:1–34:32, 2018.

6 Christophe Alias, Alain Darte, Paul Feautrier, and Laure Gonnord. Multi-dimensional rankings, program termination, and complexity bounds of flowchart programs. In *SAS*, volume 6337 of *Lecture Notes in Computer Science*, pages 117–133. Springer, 2010.

7 Christel Baier, Luca de Alfaro, Vojtech Forejt, and Marta Kwiatkowska. Model checking probabilistic systems. In *Handbook of Model Checking*, pages 963–999. Springer, 2018.

8 Christel Baier, Boudewijn R. Haverkort, Holger Hermanns, and Joost-Pieter Katoen. Performance evaluation and model checking join forces. *Commun. ACM*, 53(9):76–85, 2010.

9 Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

10 Jialu Bao, Nitesh Trivedi, Drashti Pathak, Justin Hsu, and Subhajit Roy. Data-driven invariant learning for probabilistic programs. In *CAV (1)*, volume 13371 of *Lecture Notes in Computer Science*, pages 33–54. Springer, 2022.

11 Kevin Batz, Mingshuai Chen, Sebastian Junges, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. Probabilistic program verification via inductive synthesis of inductive invariants. In *TACAS (2)*, volume 13994 of *Lecture Notes in Computer Science*, pages 410–429. Springer, 2023.

12 D. P. Bertsekas and S. E. Shreve. *Stochastic optimal control: The discrete-time case*. Athena Scientific, 1996.

13 Aleksandar Chakarov and Sriram Sankaranarayanan. Probabilistic program analysis with martingales. In *CAV*, volume 8044 of *Lecture Notes in Computer Science*, pages 511–526. Springer, 2013.

14 Ya-Chien Chang, Nima Roohi, and Sicun Gao. Neural Lyapunov control. In *NeurIPS*, pages 3240–3249, 2019.

15 Krishnendu Chatterjee, Hongfei Fu, and Amir Kafshdar Goharshady. Termination analysis of probabilistic programs through positivstellensatz's. In *CAV (1)*, volume 9779 of *Lecture Notes in Computer Science*, pages 3–22. Springer, 2016.

16 Krishnendu Chatterjee, Hongfei Fu, Petr Novotný, and Rouzbeh Hasheminezhad. Algorithmic analysis of qualitative and quantitative termination problems for affine probabilistic programs. *ACM Trans. Program. Lang. Syst.*, 40(2):7:1–7:45, 2018.

17 Krishnendu Chatterjee, Amir Kafshdar Goharshady, Tobias Meggendorfer, and Đorđe Žikelić. Sound and complete certificates for quantitative termination analysis of probabilistic programs. In *CAV (1)*, volume 13371 of *Lecture Notes in Computer Science*, pages 55–78. Springer, 2022.

18 Krishnendu Chatterjee, Thomas A. Henzinger, Mathias Lechner, and Đorđe Žikelić. A learner-verifier framework for neural network controllers and certificates of stochastic systems. In *TACAS (1)*, volume 13993 of *Lecture Notes in Computer Science*, pages 3–25. Springer, 2023.

19 Krishnendu Chatterjee, Petr Novotný, and Đorđe Žikelić. Stochastic invariants for probabilistic termination. In *POPL*, pages 145–160. ACM, 2017.

20 Fredrik Dahlqvist and Alexandra Silva. Semantics of probabilistic programming: A gentle introduction. In Gilles Barthe, Joost-Pieter Katoen, and Alexandra Silva, editors, *Foundations of Probabilistic Programming*, pages 1–42. Cambridge University Press, 2020.

21 Leonardo Mendonça de Moura and Nikolaj S. Bjørner. Z3: an efficient SMT solver. In *TACAS*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008.

22 Vojtech Forejt, Marta Z. Kwiatkowska, Gethin Norman, David Parker, and Hongyang Qu. Quantitative multi-objective verification for probabilistic systems. In *TACAS*, volume 6605 of *Lecture Notes in Computer Science*, pages 112–127. Springer, 2011.

23 Hongfei Fu and Krishnendu Chatterjee. Termination of nondeterministic probabilistic programs. In *VMCAI*, volume 11388 of *LNCS*, pages 468–490. Springer, 2019.

24 Timon Gehr, Sasa Misailovic, and Martin T. Vechev. PSI: exact symbolic inference for probabilistic programs. In *CAV (1)*, volume 9779 of *Lecture Notes in Computer Science*, pages 62–83. Springer, 2016.

25 Timon Gehr, Samuel Steffen, and Martin T. Vechev. λPSI: exact inference for higher-order probabilistic programs. In *PLDI*, pages 883–897. ACM, 2020.

26 Mirco Giacobbe, Daniel Kroening, and Julian Parsert. Neural termination analysis. In *ESEC/SIGSOFT FSE*, pages 633–645. ACM, 2022.

27 Andrew D. Gordon, Thomas A. Henzinger, Aditya V. Nori, and Sriram K. Rajamani. Probabilistic programming. In *FOSE*, pages 167–181. ACM, 2014.

28 O. Hernández-Lerma and J. B. Lasserre. *Discrete-time Markov control processes*, volume 30 of *Applications of Mathematics*. Springer-Verlag, 1996.

29 Mingzhang Huang, Hongfei Fu, Krishnendu Chatterjee, and Amir Kafshdar Goharshady. Modular verification for almost-sure termination of probabilistic programs. *Proc. ACM Program. Lang.*, 3(OOPSLA):129:1–129:29, 2019.

30 O. Kallenberg. *Foundations of modern probability*. Springer Science & Business Media, 2006.

31 Dexter Kozen. Semantics of probabilistic programs. *J. Comput. Syst. Sci.*, 22(3):328–350, 1981.

32 Marta Z. Kwiatkowska. Quantitative verification: models techniques and tools. In *ESEC/SIGSOFT FSE*, pages 449–458. ACM, 2007.

33 Marta Z. Kwiatkowska, Gethin Norman, and David Parker. Quantitative analysis with the probabilistic model checker PRISM. In *QAPL*, volume 153 of *Electronic Notes in Theoretical Computer Science*, pages 5–31. Elsevier, 2005.

34 Mathias Lechner, Đorđe Žikelić, Krishnendu Chatterjee, and Thomas A. Henzinger. Stability verification in stochastic control systems via neural network supermartingales. In *AAAI*, pages 7326–7336. AAAI Press, 2022.

35 Frederik Baymler Mathiesen, Simeon Craig Calvert, and Luca Laurenti. Safety certification for stochastic systems via neural barrier functions. *IEEE Control. Syst. Lett.*, 7:973–978, 2023.

36 Annabelle McIver and Carroll Morgan. Games, probability and the quantitative μ-calculus qmμ. In *LPAR*, volume 2514 of *Lecture Notes in Computer Science*, pages 292–310. Springer, 2002.

37 Annabelle McIver and Carroll Morgan. *Abstraction, Refinement and Proof for Probabilistic Systems*. Monographs in Computer Science. Springer, 2005.

38 Carroll Morgan, Annabelle McIver, and Karen Seidel. Probabilistic predicate transformers. *ACM Trans. Program. Lang. Syst.*, 18(3):325–353, 1996.

39 Van Chan Ngo, Quentin Carbonneaux, and Jan Hoffmann. Bounded expectations: resource analysis for probabilistic programs. In *PLDI*, pages 496–512. ACM, 2018.

40 Andrea Peruffo, Daniele Ahmed, and Alessandro Abate. Automated and formal synthesis of neural barrier certificates for dynamical models. In *TACAS (1)*, volume 12651 of *Lecture Notes in Computer Science*, pages 370–388. Springer, 2021.

41 Armando Solar-Lezama. *Program Synthesis by Sketching*. PhD thesis, University of California at Berkeley, USA, 2008.

42 Armando Solar-Lezama, Liviu Tancau, Rastislav Bodík, Sanjit A. Seshia, and Vijay A. Saraswat. Combinatorial sketching for finite programs. In *ASPLOS*, pages 404–415. ACM, 2006.

43 Toru Takisaka, Yuichiro Oyabu, Natsuki Urabe, and Ichiro Hasuo. Ranking and repulsing supermartingales for reachability in randomized programs. *ACM Trans. Program. Lang. Syst.*, 43(2):5:1–5:46, 2021.

44 Ilya Tkachev, Alexandru Mereacre, Joost-Pieter Katoen, and Alessandro Abate. Quantitative model-checking of controlled discrete-time markov processes. *Inf. Comput.*, 253:1–35, 2017.

45 Franck van Breugel and James Worrell. Towards quantitative verification of probabilistic transition systems. In *ICALP*, volume 2076 of *Lecture Notes in Computer Science*, pages 421–432. Springer, 2001.

46 Di Wang, Jan Hoffmann, and Thomas W. Reps. Central moment analysis for cost accumulators in probabilistic programs. In *PLDI*, pages 559–573. ACM, 2021.

47 Peixin Wang, Hongfei Fu, Amir Kafshdar Goharshady, Krishnendu Chatterjee, Xudong Qin, and Wenjun Shi. Cost analysis of nondeterministic probabilistic programs. In *PLDI*, pages 204–220. ACM, 2019.