TUM School of Computation, Information and Technology
Technische Universität München

Formally Verified Solution Methods for Markov Decision Processes

Maximilian Schäffeler

Vollständiger Abdruck der von der TUM School of Computation, Information and Technology der Technischen Universität München zur Erlangung eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitz:

Prof. Dr.-Ing. Matthias Althoff

Prüfende der Dissertation:

1. Prof. Tobias Nipkow, Ph.D.
2. Prof. Dr. Jan Křetínský

Die Dissertation wurde am 15.04.2025 bei der Technischen Universität München eingereicht und durch die TUM School of Computation, Information and Technology am 30.10.2025 angenommen.

Abstract

Markov decision processes (MDPs) are a fundamental model for decision making under uncertainty. They are widely used in robotics, artificial intelligence, probabilistic model checking, and operations research. Because MDPs are often applied to model safety-critical real-world systems, it is essential to ensure that the algorithms and implementations employed are trustworthy. In this thesis, we present a suite of formally verified algorithms for MDPs in probabilistic planning and model checking contexts. We formalize and verify these algorithms using the proof assistant Isabelle/HOL, focusing on formal correctness guarantees and practically efficient implementations.

In probabilistic planning, the objective is to determine a policy—a mapping from states to actions—that maximizes the expected cumulative reward accrued over time. We provide verified implementations of classical dynamic programming algorithms for this setting, including value iteration and policy iteration variants. To scale to larger systems, we extend our formalization to MDPs with compact representations, verifying the approximate policy iteration algorithm for factored MDPs. As part of this work, we develop a formally verified certificate checker for linear programming solutions, which we integrate with unverified solvers operating over exact arithmetic.

MDPs are also a prominent model in probabilistic model checking, where the goal is to check whether a probabilistic system satisfies a given logical specification of correctness. In this context, we formalize the interval iteration algorithm to derive an efficient floating-point implementation that ensures soundness through directed rounding modes. Furthermore, we use Isabelle/HOL to develop a certificate checker for MDP reachability and reward problems. The checker validates results computed by the unverified probabilistic model checker Storm.

All algorithms we verify in this dissertation are executable. Our implementations rely on either exact arithmetic or directed rounding to ensure formal correctness guarantees. We conduct an experimental evaluation on standard benchmarks to demonstrate their practicality. Our contributions demonstrate how diverse approaches to code generation and interactive theorem proving can be combined to produce efficient implementations with formal guarantees, thereby advancing the trustworthiness of MDP-based computations.

Zusammenfassung

Markow-Entscheidungsprozesse (MDPs) sind ein wichtiges Modell zur Entscheidungsfindung unter Unsicherheit. Sie finden breite Anwendung in der Robotik, bei künstlicher Intelligenz sowie im probabilistischen Model Checking und Operations Research. Da MDPs oft zur Modellierung sicherheitskritischer Systeme eingesetzt werden, ist es essenziell, dass die verwendeten Algorithmen und Implementierungen vertrauenswürdig sind. Diese Dissertation präsentiert eine Sammlung formal verifizierter MDP-Algorithmen für probabilistisches Planen und probabilistisches Model Checking. Die Algorithmen wurden mithilfe des Beweisassistenten Isabelle/HOL formalisiert und korrekt bewiesen, mit einem Fokus sowohl auf formalen Korrektheitsgarantien als auch auf praktischer Effizienz.

Bei probabilistischen Planungsproblemen besteht das Ziel darin, eine Policy, d.h. eine Zuordnung von Zuständen zu Aktionen, zu bestimmen, die die erwartete Summe der Belohnungen maximiert. Für dieses Szenario entwickeln wir verifizierte Implementierungen klassischer Algorithmen für dynamisches Programmieren, darunter Varianten des Value-Iteration-Algorithmus und des Policy-Iteration-Algorithmus. Um auf größere Systeme zu skalieren, erweitern wir unsere Formalisierung auf MDPs mit kompakter Darstellung und verifizieren eine approximative Variante der Policy-Iteration für faktorisierte MDPs. Dabei entwickeln wir einen formal verifizierten Zertifikatsprüfer für lineare Programme, der unverifizierte Optimierungsprogramme integriert, die mit exakter Arithmetik arbeiten.

MDPs sind ein zentrales Modell im probabilistischen Model Checking, wo überprüft wird, ob ein probabilistisches System eine gegebene logische Spezifikation erfüllt. Für diese Fragestellung formalisieren wir den Intervall-Iteration-Algorithmus, der als Basis für eine effiziente Gleitkomma-Implementierung dient, bei der gerichtete Rundungsmodi die Korrektheit gewährleisten. Zudem entwickeln wir mithilfe von Isabelle/HOL einen Zertifikatsprüfer für Erreichbarkeits- und Belohnungsprobleme in MDPs, der Ergebnisse des probabilistischen Model Checkers Storm validiert.

Alle in dieser Dissertation verifizierten Algorithmen sind ausführbar. Die Implementierungen setzen dabei entweder auf exakte Arithmetik oder gerichtete Rundungsmodi, um formale Korrektheit sicherzustellen. Experimentelle Evaluierungen auf Standard-Benchmarks zeigen, dass die Programme in der Praxis einsetzbar sind. Diese Beiträge demonstrieren, wie verschiedene Methoden zur Programmverifikation mit interaktiven Theorembeweisern kombiniert werden können, um effiziente Implementierungen mit formalen Garantien zu kombinieren und somit letztlich die Vertrauenswürdigkeit von MDP-Algorithmen zu erhöhen.

Acknowledgments

I am grateful for the support I received from many people throughout the past three and a half years. During this time, I had the privilege of being funded by the DFG Research Training Group ConVeY. I am grateful for the funding and the opportunity to work with a diverse group of researchers from multiple disciplines and universities. The ConVeY retreats and workshops were always fun and a great place to exchange ideas with other doctoral researchers and our principal investigators.

Over time, the chair for Logic and Verification at the Technical University of Munich has become a second home. I will always remember my former colleagues Fabian, Gabriel, Jonas, Katharina, Kevin, Lukas, Manuel, and Simon for the countless eventful trips to the Mensa and the various lunch debates that broadened my horizons. Originally the advisor of my Master's thesis, my mentor Mohammad Abdulaziz spent countless hours with me discussing research ideas and life far beyond research. Thank you for your guidance and support, especially during difficult times. I am deeply grateful to my supervisor, Tobias Nipkow, for the opportunity to work at your chair and for the academic freedom you provided. Thank you for introducing me to the world of formal methods and verification early on through the course "Perlen der Informatik". Our team assistant, Helma Piller, was always approachable and helped me navigate the university's administrative landscape. Finally, I would like to thank Jan Křetínský and Matthias Althoff for kindly agreeing to serve as second examiner and chair, respectively.

I would like to thank all my collaborators and fellow researchers with whom I had the pleasure of working: Michael Akintunde, Marvin Brieger, Krishnendu Chatterjee, Arnd Hartmanns, Bram Kohlen, Hanna Krasowski, Peter Lammich, Tim Quatmann, Maximilian Weininger and Tobias Winkler. Many students at TUM also contributed through thesis projects, practical courses, and seminars. Thank you!

Finally, I am deeply indebted to my family and all my friends for their sustained patience and encouragement. Thank you for all the trips and vacations that helped me recharge my batteries—from cycling throughout Europe, traversing glaciers in the Alps and beyond. I am especially grateful to my brother, soulmate and flatmate Jakob, my parents Birgit and Robert, and my grandmother Elisabeth for their constant support.

I am grateful to Jakob and Mohammad for their valuable feedback on earlier drafts of this thesis. I also acknowledge the use of writing tools that supported the editing process.

Contents

Index of Publications

This cumulative dissertation is based on the four publications listed below, all of which are classified as *Core Publications*. In each case, I am either the sole or shared first author. Chapter 6 provides summaries of these works and outlines my own contributions to each publication. The original publications are included in the appendix.

Core Publication 1 (Factor of core: 1, sole first author)
Maximilian Schäffeler and Mohammad Abdulaziz. "Formally Verified Solution Methods for Markov Decision Processes." In: *AAAI Conference on Artificial Intelligence*. Vol. 37. 12. AAAI Press, 2023, pp. 15073–15081. DOI: 10.1609/aaai.v37i12.26759

Core Publication 2 (Factor of core: 1, sole first author)
Maximilian Schäffeler and Mohammad Abdulaziz. "Formally Verified Approximate Policy Iteration." In: *AAAI Conference on Artificial Intelligence*. Vol. 39. AAAI Press, 2025. To appear.

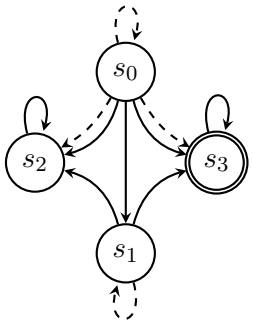
Core Publication 3 (Factor of core: 0.5, two first authors)
Bram Kohlen, Maximilian Schäffeler, Mohammad Abdulaziz, Arnd Hartmanns, and Peter Lammich. "A Formally Verified IEEE 754 Floating-Point Implementation of Interval Iteration for MDPs." In: *Computer Aided Verification*. Lecture Notes in Computer Science. Springer, 2025. To appear.

Core Publication 4 (Factor of core: 0.17, six first authors)
Krishnendu Chatterjee, Tim Quatmann, Maximilian Schäffeler, Maximilian Weininger, Tobias Winkler, and Daniel Zilken. "Fixed Point Certificates for Reachability and Expected Rewards in MDPs." In: *Tools and Algorithms for the Construction and Analysis of Systems*. Lecture Notes in Computer Science. Springer, 2025. To appear.

1 Introduction

Markov Decision Processes (MDPs) [134] are a foundational model for sequential decision-making under uncertainty. Intuitively, MDPs describe scenarios where an *agent* interacts with an environment by performing actions. Each action probabilistically determines the system's transition to a successor state and may incur a cost or yield a reward. The behavior of the agent is determined by a *policy* (or *strategy*) that prescribes which action to take in each state. MDPs are a simple yet expressive model, hence they are popular across several research domains, including *probabilistic planning* [134], *reinforcement learning* (RL) [148], and *probabilistic model checking* (PMC) [17]. They have found applications in robotics, inventory management, and other real-world systems characterized by uncertainty [159]. Example 1 illustrates the concept of MDPs using a simple example with four states and two actions.

Example 1. *An MDP with four states, a maximum of two actions per state (solid and dashed edges, respectively), uniform transition probabilities, and a target set* $T = \{s_3\}$*. The maximum reachability probability from* s_0 *to* T *is* $\mathbb{P}^{\max}_{s_0}(\Diamond T) = \frac{1}{2}$*: it may be achieved by choosing the solid action at every step. Conversely, we need to choose the solid action at* s_0 *and the dashed action at* s_1 *to attain the minimum reachability probability* $\mathbb{P}^{\min}_{s_0}(\Diamond T) = \frac{1}{3}$*.*



In probabilistic planning and model checking, we are often interested in answers to questions such as: 1) *"What is the probability that a network file transfer will eventually succeed?"* [44], 2) *"Which maintenance strategy minimizes service outages within a given budget?"* [140, 141], or 3) *"What is the optimal age to replace vehicles in a fleet, balancing repair and acquisition costs?"* [78]. These practical concerns may be abstracted to quantities such as the *expected cumulative reward* of a policy, or the *maximum probability* of reaching a set of states from a given initial state. To determine such quantities efficiently and automatically on MDP models, a variety of solution methods have been developed. Classical dynamic programming algorithms, most notably *value iteration* and *policy iteration*, are widely used to compute optimal policies. Additionally, many problems can be solved by reformulating them as *linear programs* (LPs) [72, 134].

Trustworthiness of MDP Algorithms Given that MDPs are frequently used to model safety-critical systems, it is essential to ensure that the solution methods and their implementations are trustworthy. In recent years, however, several issues have emerged that threaten the validity of results produced by PMC and probabilistic planning tools. These soundness problems can be broadly classified into four categories.

First, *implementation bugs* are common in large unverified software systems, including modern probabilistic model checkers. For instance, the implementation of the *interval iteration* algorithm for expected rewards in the Modest toolset diverges on a small percentage of instances [18].

Second, *algorithmic unsoundness* results from using optimizations, heuristics, or even core algorithms that are not sound in general. A notable example is the value iteration algorithm, which has been shown to not be applicable to PMC [64]. This discovery led to the design of several sound alternatives, such as interval iteration [64, 73, 136]. However, even these can have subtle issues, as demonstrated by the pseudocode for the *sound value iteration* algorithm in the original presentation [136]. It contains a problem that may only be observed on a single instance of the *Quantitative Verification Benchmark Set* (QVBS) [74].

Third, many implementations rely on *floating-point arithmetic* to achieve state-of-the-art performance, which introduces *rounding errors* [70]. Finally, errors in third-party libraries or tools like commercial LP solvers may be a source of failure. These errors are often difficult to detect and resolve but can significantly affect the correctness of results [72].

Correct-by-Construction Implementations The issues outlined above highlight the need for formal specifications of MDP algorithms, accompanied by machine-checked correctness proofs; a domain which was not sufficiently explored in the past. However, as discussed previously, even correct algorithms may be incorrectly implemented. Current state-of-the-art tools for MDP analysis—such as planning tools or probabilistic model checkers—are typically implemented manually and remain unverified, relying on pen-and-paper proofs or extensive testing [71, 79, 108]. This underscores the importance of extending our focus to correct-by-construction implementations.

Various techniques exist to improve the trustworthiness of software, including pen-and-paper proofs, testing, and static analysis tools. Each of these, however, has inherent limitations: pen-and-paper proofs are error-prone and often do not scale well to complex algorithms; testing can only ever cover a subset of all possible inputs, making it unsuitable for ensuring general correctness; and static analysis tools are constrained in the specifications they can express and verify. Also, we still depend on the correctness of their potentially complex implementations.

One promising methodology for developing trustworthy software is the use of *interactive theorem provers* (ITPs). ITPs enable users to build formal mathematical libraries and, in many cases, support the generation of executable code from verified definitions [65]. Notable examples of projects that use ITPs for software verification include

the formally verified operating system kernel seL4 [102], a verified compiler for C programs [111], and a verified non-probabilistic model checker [59].

In this work, we present a suite of verified algorithms for probabilistic planning and model checking developed within the ITP Isabelle/HOL [124]. MDPs have already drawn attention in the ITP community: previously Hölzl formalized probability theory and MDPs in Isabelle/HOL [83], while others have verified dynamic programming algorithms for MDPs in Rocq [155] and Isabelle/HOL [39].

Challenges Despite prior verification efforts of algorithms for probabilistic planning and model checking in ITPs, few have resulted in efficient implementations that demonstrate practical utility. Developing verified software is a demanding task, particularly in the context of MDPs.

At an *abstract level*, MDP analysis requires combining a range of non-trivial concepts from mathematics, including probability theory, optimization, limits, and graph theory. To make such formalizations feasible, an ITP must provide a large, integrated library of formal mathematics. In this regard, Isabelle is well-suited: it provides a rich library of formalized mathematics and theoretical computer science in the *Archive of Formal Proofs*[1] (AFP) and integrates with automated theorem provers via Sledgehammer [131].

At the *implementation level*, one major challenge lies in selecting appropriate numerical representations. MDP algorithms involve computations on probabilities, which are usually implemented as *floating-point arithmetic* for performance. However, floating-point computations introduce rounding errors, making it difficult to obtain provably correct results. We consider two viable approaches to mitigate this issue. The first is to use *exact arithmetic* on rational numbers, which eliminates rounding errors but may lead to a significant loss in performance. The second is to retain efficient floating-point arithmetic, but modify the computations in a way that they produce sound, conservative approximations of the correct result. Techniques such as interval arithmetic or the use of directed rounding modes fall into this category [70].

More broadly, generating efficient executable code from formalizations remains a non-trivial task. Several approaches exist, each with different tradeoffs in performance and verification effort. These include Isabelle/HOL's built-in code generation facilities and the *Isabelle Refinement Framework* (IRF) [110].

Another challenge concerns the architecture of the verified software system. One option is to derive an implementation directly from an Isabelle formalization. Alternatively, one could verify existing implementations using frameworks such as *AutoCorres* [126]. A further promising approach is the use of *certifying algorithms* and *certificate checking*. With this paradigm, the algorithm produces a certificate of its result—a small, easily verifiable proof of its result—that can be checked independently by a formally verified program. This technique is already in use in competitions for SAT solving and software verification [19, 29]. However, existing work on certifying algorithms for MDPs [60, 83, 95] has several limitations that hinder broader adoption. These are discussed in detail in Section 3.3.

[1] `https://www.isa-afp.org`

Contributions The main contribution of our work is the development of practical solutions to all the challenges outlined above. In particular, we explore and contrast different approaches and methodologies for algorithm verification and code generation in Isabelle/HOL. We examine the trade-offs between certificate checking and fully verified implementations, develop imperative implementations, as well as functional programming implementations of MDP algorithms, and assess the feasibility of exact arithmetic versus floating-point arithmetic with directed rounding. Our detailed contributions are as follows:

- We use Isabelle/HOL to develop verified, executable algorithms for probabilistic planning on explicitly represented MDPs in *Core Publication 1* [5]. Specifically, we implement classical *dynamic programming algorithms*, including value iteration and policy iteration. Our implementations rely on exact arithmetic to guarantee correctness. To achieve practical performance, we propose a hybrid approach that integrates unverified floating-point solvers with our work.

- In *Core Publication 2* [144], we formally verify an implementation of *approximate policy iteration* for solving MDPs with compact representations. We demonstrate experimentally that this verified implementation scales to larger systems compared to the above work on explicitly represented MDPs. As part of this effort, we develop a certificate checker that validates linear programming solutions. Our work on probabilistic planning is incorporated into the *FormPlan* toolset of formally verified planning software.[2]

- In *Core Publication 3* [103], we present a formalization of the *interval iteration* algorithm for MDP reachability analysis. This algorithm can be combined with directed rounding of floating-point computations to produce an implementation that is both sound and efficient. We formally verify soundness and convergence in Isabelle/HOL, and derive a sound floating-point implementation using the *Isabelle Refinement Framework* [110].

- As part of *Core Publication 4* [38], we develop a *verified certificate checker* for reachability and reward properties for MDPs, using a novel certificate format based on fixed points [38]. This checker is used to formally certify reference results from the *Quantitative Verification Benchmark Set* [74].

Outline The remainder of the thesis is structured as follows: Chapter 2 introduces MDPs and the proof assistant Isabelle/HOL. Chapter 3 reviews related work in the areas of probabilistic verification and formal methods. Chapter 4 presents the core contributions of the included publications and discusses how they relate to one another. Chapter 5 concludes the thesis with a summary of our findings and outlines directions for future research. Finally, Chapter 6 provides summaries of the individual publications that make up this dissertation and lists the specific contributions of each author.

[2] `https://formplan.github.io/`

2 Background

In this chapter, we introduce the fundamental concepts that support the formalization of algorithms for MDPs. We begin by presenting the basic concepts of MDPs, the central model studied in this thesis. We give a formal definition of MDPs and the notion of *policies*. Next, we describe the key tool underlying our methodology for software verification: interactive theorem provers, in particular the proof assistant Isabelle/HOL.

2.1 Markov Decision Processes

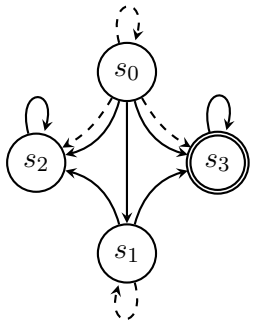
Markov Decision Processes (MDPs) model systems that evolve over time, where the dynamics are probabilistic but can be influenced by an agent interacting with the system. At each step, the agent observes the current state of the environment and chooses an action from a set of available options. Executing the selected action leads to a stochastic transition to a successor state, determined by both the current state and the chosen action. In some settings, the agent also receives a reward based on the action taken. The reward quantifies how desirable the resulting behavior is.

Figure 2.1 presents a small example MDP to illustrate the concepts introduced in this section. We formally define finite MDPs, as used throughout this thesis, in Definition 1. For a more comprehensive introduction to MDPs, we refer the reader to standard literature [17, 134, 148].

Definition 1. *A finite MDP is a tuple* $\mathcal{M} = (S, Act, P)$ *where* S *is a finite set of* states*,* Act *is a finite set of* actions*, and* $P\colon S \times Act \times S \to [0,1]$ *is a* transition probability function *that satisfies* $\sum_{s' \in S} P(s,a,s') \in \{0,1\}$ *for all* $s \in S$ *and* $a \in Act$*. For every* $s \in S$*, the set of* enabled *actions* $a \in Act$ *for which the above sum equals* 1 *is written* $Act(s)$*. It is required that* $Act(s) \neq \varnothing$ *for all* $s \in S$*. A discrete-time* Markov chain *is a special case of MDP in which each state has exactly one action, i.e.,* $|Act(s)| = 1$ *for all* $s \in S$*.*

Policies The set of action preferences for each state is collectively captured by a *policy*—in its simplest form, a function mapping states to actions. Depending on the application domain, policies are also referred to as strategies, schedulers, controllers, or adversaries. Various classes of policies exist, differing in the amount of information they use when selecting actions.

A *memoryless* and *deterministic* policy for an MDP $\mathcal{M} = (S, Act, P)$ is a function $\sigma\colon S \to Act$ such that for all $s \in S$, we have $\sigma(s) \in Act(s)$. Such a policy is also



Let $\mathcal{M} = (S, Act, P)$ with $S = \{s_0, s_1, s_2, s_3\}$,
$Act = \{a_0, a_1\}$, (a_0 is solid, a_1 is dashed)
$P(s_0, a_0, s_1) = P(s_0, a_0, s_2) = P(s_0, a_0, s_3) = \frac{1}{3}$,
$P(s_1, a_0, s_1) = 1, P(s_1, a_1, s_2) = P(s_1, a_1, s_3) = \frac{1}{2}$, etc.
Define the policy $\sigma : S \to Act$ with
$\sigma(s_0) = \sigma(s_2) = \sigma(s_3) = a_0, \sigma(s_1) = a_1$, then
$\mathbb{P}^{\min}_{s_0}(\Diamond T) = \mathbb{P}^{\sigma}_{s_0}(\Diamond T) = \frac{1}{3}$.

Figure 2.1: On the left, we show an MDP with four states and a maximum of two actions per state (solid and dashed edges, respectively). The transition probabilities are uniform, e.g. each of the solid outgoing edges from s_0 has a probability of $\frac{1}{3}$. We also define a target set $T = \{s_3\}$. On the right, we show a definition of the MDP on the left and a policy σ that achieves the minimum reachability probability from s_0 to T.

referred to as a *decision rule* [134]. Applying σ to $\mathcal{M}$ yields the *induced* Markov chain $\mathcal{M}^\sigma = (S, Act, P^\sigma)$, which intuitively retains only the actions chosen by the policy.

More expressive policies can incorporate additional information. *History-dependent* policies take into account the full history of states and actions observed by the agent. A special case of history-dependent policies are *Markovian* policies, which depend only on the current state and the number of steps taken since the beginning. *Stochastic* policies generalize the decision process further by choosing actions according to a probability distribution over the enabled actions. A central question is whether a particular problem for MDPs admits an optimal solution that is memoryless and deterministic. All problems we study admit policies that are either memoryless deterministic or Markovian.

Markov Decision Problems When using MDPs to model and analyze systems, we typically pair the MDP with a specific objective, for example, determining the minimum probability of reaching a target state within a number of steps, or identifying an optimal policy to maximize cumulative rewards. We refer to the combination of an MDP and an objective as a *Markov decision problem* [134]. Two fundamental classes of objectives are *reachability* and *expected reward* objectives. These are defined in detail in Section 4.1 and Section 4.2, where we also discuss our corresponding contributions.

Given a reachability objective, we aim to find a policy that maximizes (or minimizes) the probability of reaching a designated set T of target states in the long run. The maximum reachability probability to reach T from an initial state $s \in S$ is denoted by $\mathbb{P}^{\max}_s(\Diamond T)$. Analogously, the maximum expected reward collected before reaching T from s is written as $\mathbb{E}^{\max}_s(\Diamond T)$. Minimum reachability probabilities and expected rewards are defined similarly. In the context of *probabilistic planning*, we omit the target set but introduce a *discount factor*, which encourages the agent to favor immediate rewards over delayed ones.

2.2 Isabelle/HOL

We conduct our formalization project to verify algorithms for MDPs using the *interactive theorem prover* (ITP) Isabelle/HOL. The ITP is used throughout the entire process: we define the algorithms formally, prove correctness properties, and refine them into verified implementations—all within the proof assistant.

An ITP is a software tool that assists users in defining formal mathematical objects and proving their properties within a logical framework. Unlike *automated theorem provers* (ATPs), where proofs are generated automatically, ITPs require users to construct proofs interactively. This involves specifying proof steps, applying logical rules, and breaking down complex problems into smaller lemmas. The system then checks these proofs for correctness, verifying them down to the axioms of the underlying logic.

Proof assistants are widely used in formal verification, mathematics, and computer science to establish trust in pen-and-paper proofs, algorithms [123], software [59], and hardware [142]. Prominent interactive theorem provers include tools such as Lean [49], Rocq [28], as well as the system used in this work, Isabelle/HOL [124]. Isabelle/HOL is an ITP based on *higher-order logic* (HOL), which can be seen as a combination of functional programming with logic. We choose Isabelle/HOL for our work due to its extensive libraries, powerful automated proof methods, and mature facilities for generating verified executable code:

- The *Archive of Formal Proofs* (AFP) is a curated collection of Isabelle theories and proofs that are not part of the core Isabelle distribution. Together, the AFP and the Isabelle distribution provide a large body of reusable libraries of formal mathematics, including libraries on topics such as analysis and probability theory. These libraries form the foundation for our formalization projects.

- A key strength of the Isabelle/HOL system lies in its powerful automated proof methods. The Sledgehammer tool integrates the proof assistant with external ATPs and SMT solvers such as Vampire and Z3. Sledgehammer selects relevant facts from the proof context and passes them to these external provers. If a proof is found, Sledgehammer reconstructs it internally. Additionally, Isabelle/HOL provides several powerful proof methods such as the *simplifier*. These automate common proof patterns like term rewriting, induction, etc.

- Formal definitions in Isabelle/HOL can be turned into executable code. The code generation functionality allows verified algorithms to be exported as code in languages such as Haskell, Scala, or Standard ML [65]. Alternatively, the *Isabelle Refinement Framework* offers a structured and partially automated methodology for refining abstract specifications into efficient implementations [110].

Locales One particular feature of Isabelle/HOL we wish to highlight is its support for *locales* [99]. Locales provide formal contexts that facilitate the modularization and reusability of formal proof developments. Our Isabelle/HOL developments make heavy use of locales to structure and organize the verification of MDP algorithms.

```
1  locale MDP =                                  MDP with states of type 's, actions of type 'a
2    fixes   Act :: 's ⇒ 'a set                  set of enabled actions per state
3      and   P :: 's × 'a ⇒ 's pmf                transition probabilities
4    assumes
5      Act_ne: ⋀s. Act s ≠ ∅                     every state has enabled actions
```

Figure 2.2: A simplified version of our locale definition for MDPs in Isabelle/HOL [145]. The names of the locale parameters have been adapted to match the notation used throughout this thesis.

A locale defines a set of fixed constants and assumptions, which may be thought of as a parameterized context for developing definitions and theorems. For example, in the formal definition of an MDP shown in Figure 2.2, the corresponding locale fixes a type $'s$ to represent the state space, a function defining the enabled actions of type $'a$ for each state, and an additional function mapping state-action pairs to transition probabilities. The locale may also contain assumptions over these constants, for instance, that every state has at least one enabled action.

Locales support nesting, complex hierarchies and inheritance, as well as instantiation. When a locale is instantiated with concrete types and functions (e.g. a specific MDP), all theorems proven in the locale context become available for the instantiated object. The instantiation process requires the user to discharge the locale assumptions through suitable proofs, to show that the theorems stated in the locale context hold for the concrete instance.

3 Related Work

We provide a brief overview of the current state of the art in formalizing probability theory using ITPs, as well as the broader application of formal methods in the areas of MDPs, planning, and artificial intelligence. In addition, we examine prior work related to the specific methodologies we employed in our own work, including the verification of floating-point algorithms and certificate checking techniques. This chapter aggregates and expands upon the related work sections from our *Core Publications*.

3.1 Verification of Floating-Point Algorithms

It is widely recognized as a problem that floating-point implementations of algorithms often deviate from their corresponding mathematical models. To achieve high performance, floating-point computations—as specified by the IEEE-754 standard—use optimized, finite-precision approximations of real numbers. This introduces rounding errors, which may accumulate in complex algorithms. Moreover, extremely small or large numbers can cause underflow and overflow, respectively. The non-associativity of floating-point arithmetic also means that results can vary depending on the order of operations.

Bugs resulting from these properties have been observed in the hardware and aerospace industries [68, 153], where they can have potentially serious consequences. Due to the intricate behavior of floating-point algorithms, such bugs are difficult to reliably detect through conventional testing. This challenge has inspired a long-standing effort to apply formal methods to the verification of floating-point algorithms in systems such as Z [22], HOL Light [67, 68, 69], PVS [34, 119], and Rocq [33, 48].

In Rocq (previously known as Coq), a formal specification of the IEEE-754 standard has been developed, enabling the verified compilation of floating-point operations [32]. This specification has also served as the basis for a framework to verify C programs involving floating-point computations [137]. Similarly, a specification of the IEEE-754 standard exists in Isabelle/HOL [167], which our work builds on. Finally, in ACL2, hardware implementations of floating-point operations have also been verified [142].

Most prior work has focused on verifying the correctness of basic floating-point operations and algorithms. In contrast, in *Core Publication 3* [103], we aim to prove the correctness of a complex algorithm at the level of real-number computations. These computations are then systematically transformed into floating-point operations using directed rounding modes. Rounding consistently upward or downward, depending on the context, guarantees that the floating-point result is a lower or upper bound of the

real-number result—without requiring intricate manual proofs of floating-point error bounds.

A related line of research aims to enhance the reliability of floating-point implementations by providing sound error bounds. Tools such as PRECiSA [152], FPTaylor [147], Real2Float [113], and Fluctuat [62] propagate floating-point errors with static analysis to determine the worst-case round-off error. Unlike these approaches, our methodology does not provide explicit error bounds. However, these tools typically offer limited support for programs with complex, nested control flow.

The static analysis tool Astrée [43], used in the aviation and automotive industry, checks the absence of runtime errors in programs and supports floating-point arithmetic. However, being an automatic tool based on abstract interpretation, it cannot verify arbitrary correctness properties. Frama-C [101] provides similar functionality and also supports deductive verification. Still, the scope of supported properties is limited, for example, it may be used to show that program outputs lie within a specific interval [104].

Other deductive verification tools, such as KeY, offer comparable capabilities. Key has been used to verify the absence of exceptional floating-point values such as *NaN* and *infinity* [2]. An alternative methodology is presented by Nguyen and Marché [122], who use the Why platform to analyze behavioral properties directly at the level of compiled assembly code.

3.2 Certificate Checking

Certifying algorithms [115] refer to a class of algorithms that not only produce a solution but also provide additional information that serves as a proof of the solution's correctness. The goal is for these proofs—called *certificates*—to be validated through a simple and efficient certificate checking procedure. The certificates could be verified by independent third parties, and even by formally verified implementations. In the simplest cases, such as sorting a list, the result is already considerably easier to check than to compute, and thus no additional information is necessary for verification. While the general technique of *certifying algorithms* has been used for decades, the term itself was popularized in the context of the *LEDA* library of algorithms [105].

In model checking and formal methods, certificate checking is used to increase the trust in results produced by highly optimized implementations, which often rely on complex heuristics and optimizations. Certifying algorithms have been designed for model checking of linear-time properties [66, 107, 120, 132]. In this setting, a negative result can be witnessed by a simple counterexample, whereas a positive result requires a deductive proof in a custom proof system. Similar proof systems have also been developed in classical planning to certify the unsolvability of planning problems [57, 58]. The technique of certifying algorithms has recently been extended to hardware verification and model checking [165, 166]. For related work on certifying algorithms in the context of probabilistic model checking we refer to Section 3.3.

Formally Verified Certificate Checkers The certifying algorithms presented in the previous section are not accompanied by formally verified checkers that can be used to validate their results. As a consequence, we must still rely on the correctness of the (typically simpler) certificate checker. In our work, we construct formally verified certificate checkers for Markov decision problems, presented in detail in *Core Publications 1, 2, and 4*. Formal verification of certificate checkers is a natural application of software verification using ITPs, since certificate checking algorithms are considerably easier to implement and verify than the algorithms they are designed to certify.

This approach has previously been applied to SAT and SMT problems, which are important domains for certificate checkers due to the wide range of problems that can be reduced to SAT or SMT. GRAT is a verified, high-performance certificate checker for SAT solutions, based on unsatisfiability certificates in the *LRAT* format [109]. The Rocq library *SMTCoq* [55] certifies results produced by SMT solvers. Another general problem domain is linear programming (LPs), for which we design a formally verified checker in *Core Publication 2* [144]. Prior work on LP certificate checkers in Isabelle/HOL was performed by Obua [127], whose formalization can certify arbitrary floating-point upper bounds of LP solutions. In contrast, our project can certify optimality of LP solutions, although we currently only support solvers using exact arithmetic.

Additional applications of formally verified certificate checkers include approximate model counting [149], floating-point rounding error analysis [26], certifying compilers built in Isabelle and Rocq [31], and a framework for verifying iterative implementations of certificate checkers in Isabelle/HOL [9, 126].

3.3 Reliable Probabilistic Model Checking

Probabilistic model checking (PMC) [14, 17] is a formal verification technique for analyzing systems that exhibit probabilistic behavior, such as MDPs. The properties of interest typically include the probability of satisfying linear-time properties, or the expected value of rewards. Mature probabilistic model checkers include PRISM [108], Storm [79], and mcsta [71].

Our work aims to enhance the trustworthiness of PMC results. In the past, several soundness-critical issues in PMC algorithms and their implementations have been discovered and subsequently addressed. These problems span various aspects of probabilistic model checkers, from unsound algorithms [18, 64] and numerical inaccuracies [70] to errors in third-party libraries [72]. To address these challenges, we develop formally verified software for PMC: a certificate checker, and an implementation of the interval iteration algorithm.

Several prior approaches have aimed to certify PMC results. Most closely related to our work is the technique by Hölzl [83, Sec. 4], who presents a formally verified certificate checker for reachability probabilities based on *fixed point certificates*. Fixed point certificates exploit the fact that many properties in MDPs can be expressed as the fixed points of monotonic operators. However, that work does not cover mechanisms for certificate generation and provides no experimental evaluation. Moreover, the supported properties

are limited to upper bounds on maximum and lower bounds on minimum probabilities. Our formalization, based on the theory presented in *Core Publication 4* [38], extends this approach to a broader class of properties.

Additionally, for non-probabilistic model checking, Esparza et al. [59] present an executable model checker for *linear temporal logic* (LTL) specifications that has been verified in Isabelle/HOL.

An alternative path towards trustworthy PMC is proposed by Funke, Jantsch, and Baier [60, 94], who present *Farkas Certificates* for reachability properties on MDPs without end components. Although this restriction to MDPs can be lifted by applying a certified maximal end component decomposition algorithm, doing so results in significantly more complex certificates. Furthermore, their work does not explicitly address certificate generation. While the method also supports certification of expected reward properties, it requires that the set of target states is reached almost surely. In contrast, our work on certificates [38] imposes no such structural restrictions on the MDP. Baier, Chau, and Klüppelholz [15] extend Farkas certificates to multi-objective queries that involve combinations of mean payoff or reachability objectives.

Witnessing subsystems [15, 60, 95, 160] offer yet another certificate format for PMC. In this method, small subsystems of the MDP that witness the property in question constitute the certificate. While such witnesses exist for a wide range of properties, the approach involves computationally more intensive checks than the state-local operations required for validating Farkas or fixed point certificates.

3.4 Formal Methods in Artificial Intelligence

MDPs are a foundational model in artificial intelligence (AI), extensively studied in the contexts of reinforcement learning and probabilistic planning. However, the use of ITPs and formal methods in AI has previously been more prevalent in other domains, including machine learning and classical planning.

Machine Learning For concrete neural networks, Marabou is a solver designed for neural network verification problems [161]. It translates the neural network and the property to be verified into a set of linear and non-linear constraints, which are then solved using a combination of mixed-integer linear programming and SMT solving. A typical application of Marabou is verifying robustness against adversarial attacks, i.e. ensuring the output of the network remains stable when the input is perturbed slightly. Marabou is also the default verifier for the domain-specific language Vehicle, which integrates with the ITP Agda [45].

Unlike automated tools such as Marabou, ITPs enable the verification of not only neural networks themselves but also the underlying training procedures. MLCERT is a verified system for proving generalization guarantees in machine learning, ensuring that models generalize to unseen data [13]. In a similar vein, Selsam, Liang, and Dill [146] present a methodology for verifying AI algorithm implementations in the ITP Lean. As a

case study, they build Certigrad, a framework capable of training variational autoencoders with competitive performance.

ITPs have also been used to formalize theoretical results in machine learning. For example, the superiority of deep learning over shallow learning has been formally proved in Isabelle/HOL by Bentkamp, Blanchette, and Klakow [27].

Conversely, there is ongoing research into enabling AI agents to interact with ITPs [21, 88, 93, 96, 97]. This includes work on *autoformalization*: the process of translating informal mathematical text into formal definitions and proofs [162]. Systems such as AlphaProof [61] use a combination of language models and reinforcement learning to generate formal proofs in Lean, and have been applied to solving International Mathematical Olympiad problems.

Planning Classical planning is a core domain in AI focused on computing action sequences that transition a system from an initial state to a goal state, based on a set of actions and effects. Several planning algorithms have been explored using proof assistants. For example, the ITP HOL4 was employed to detect and correct flaws in a planning algorithm [3].

In Isabelle/HOL, Abdulaziz and Kurz [5] developed planning software that reduces planning problems to SAT encodings. This system was used as a reference to uncover bugs in a state-of-the-art SAT-based planner. Beyond verified planning algorithms, Abdulaziz et al. [4, 6] also introduced formally verified plan validators, based on formal semantics of *Planning Domain Definition Language* (PDDL) fragments.

In the dependently typed programming language Agda, Hill, Komendantskaya, and Petrick [80] formalize a novel logic for reasoning about properties of plans. They formally verify that this logic is sound with respect to a formal semantics of PDDL. Furthermore, their work presents a pipeline that parses PDDL plans into Agda and automatically generates correctness proofs within the logic.

3.5 Probabilistic Analysis in Interactive Theorem Provers

Formalizing probability theory in ITPs is a well-studied, yet inherently challenging topic. Mechanizing results from probability theory is demanding for several reasons. First, formal probability theory for continuous probability spaces is based on measure theory, requiring a proof assistant that provides a robust library of results from real analysis. Second, probability theory is commonly divided into the study of *discrete* and *continuous* probability spaces. The former concerns countable outcomes such as dice rolls, while the latter deals with uncountable outcomes like real numbers. Designing a probability theory library that supports both domains effectively—while minimizing the duplication of concepts—is a nontrivial task. Third, informal treatments of probability theory often employ concise notation that hides underlying details, e.g. the definition of the measure space and proof obligations related to measurability and integrability.

Numerous ITP libraries on probability theory have addressed these challenges to varying extents. We begin with an overview of libraries for probabilistic analysis in ITPs,

followed by a discussion of work on the probabilistic analysis of algorithms and stochastic processes. Hölzl [81] provides a comprehensive account of the formalization of probability theory in ITPs up to 2013, with a particular focus on Isabelle/HOL. Our survey of the literature demonstrates that Isabelle/HOL offers a versatile and widely adopted probability theory library, which makes it a solid foundation for developing verified implementations of MDP algorithms.

3.5.1 Formal Libraries of Probability Theory

The first formal library of measure and probability theory in an ITP was developed for the Mizar system [20] by Nędzusiak [121] in 1990, and was later extended by Białas [30]. Over time, several foundational results were formalized in Mizar, including Dynkin's Lemma [116], Lebesgue integrals [56], uniform distributions [128], Kolmogorov's 0-1-law [50], the Chebyshev inequality [129], and the Borell-Cantelli Lemma [91].

Early efforts to formalize probabilistic algorithms in *higher-order logic* (HOL) were published by Hurd [89], who designed the first formal library of probability theory in HOL for the system HOL98. This work focused on verifying sampling algorithms for discrete probability distributions and culminated in a verified implementation of the probabilistic *Miller-Rabin primality test*. It was later extended to also cover sampling algorithms for continuous distributions and concepts such as expectations and variance [77]. Building on this, formalizations in the successor system HOL4 expanded to include Lebesgue integration [40], Radon-Nikodym derivatives, and results from information theory [117, 118, 135]. The HOL4 library supported applications such as formally verified *reliability analysis* [1].

In Isabelle/HOL, early work by Richter [138] involved automatically importing parts of the probability theory library from HOL4. The modern standard library for probability theory in Isabelle/HOL was developed in large parts by Hölzl [81, 84], building on prior work in HOL4. The library formalizes key concepts such as measure spaces, probability spaces, and random variables. It has since been widely adopted to formalize theorems from number theory [125], the central limit theorem [12], and a verified compiler for probability density functions [52]. Practical applications also include formalizations of the security property probabilistic noninterference [133] and formal semantics for the expected running time of probabilistic programs [82]. We discuss further applications in Section 3.5.2 and Section 3.5.3.

The system PVS [130] has been used to formalize probability and measure theory, with support for discrete, continuous and mixed random variables [46, 112]. The library also covers Markov's and Lévy's inequalities, binary product spaces, and finite families of independent random variables. It was used to prove stochastic bounds on the correctness of numerical algorithms [47].

Several libraries for probability theory have been developed in the Rocq prover [28]. A framework for analyzing randomized algorithms in Rocq has been designed by Audebaud and Paulin-Mohring [11]. However, it only supports *discrete* probability spaces. Information theory, specifically results about lossy encoding, has also been formalized in Rocq [8], though the scope remains limited to discrete distributions. More recently, mea-

sure theory has been developed within the *Mathematical Components* library [7]. The Lean prover [49], which shares a similar logical foundation with Rocq, has been applied to formalize Bayesian probability theory [154] and measure theory [158]. A library of probability theory is currently under development in Lean by the *mathlib* community [42, 114].

These foundational libraries have enabled numerous applications, including the formal analysis of randomized algorithms and stochastic processes, which we discuss next.

3.5.2 Analysis of Randomized Algorithms and Data Structures

Libraries for probability theory in ITPs have been applied to a wide range of randomized algorithms and data structures. The formal analysis of the probabilistic *Miller-Rabin primality test* in HOL98 constitutes seminal early work in the analysis of randomized algorithms using ITPs [89]. More recently, randomized quicksort and randomized binary search trees have been formally analyzed in Isabelle/HOL [51] and Rocq [150, 157]. Bloom filters have also been formalized in Rocq using the *infotheo* library, which supports reasoning about discrete probability spaces [8].

The *Guarded Command Language* (GCL) serves as a high-level formalism for specifying non-deterministic algorithms. *Probabilistic GCL* (pGCL) extends GCL with probabilistic choice. In HOL, pGCL has been formalized and applied to derive bounds on the expected running time of probabilistic programs [37, 90]. In Isabelle/HOL, these expected running times were analyzed in a framework that demonstrates the equivalence between denotational and MDP-based pGCL semantics [82]. Another independent formalization of pGCL in Isabelle/HOL has also been carried out by Cock [41].

Cryptography *Game-based proofs* are a central paradigm in cryptographic proofs. To apply this technique, the cryptographic scheme is expressed as a sequence of games between a player and an adversary. During the games, the task of the adversary is typically to extract information—such as secret keys—from the system, despite limited access and computational constraints. The security relies on the fact that the adversary's *probability* of success is negligible.

In Isabelle/HOL, the CryptHOL framework provides a structured environment for expressing security properties and conducting game-based proofs within an ITP [25]. This library has been used to verify security properties and correctness of the post-quantum cryptographic system *CRYSTALS-KYBER* [106]. Likewise, the EasyCrypt proof assistant, which is tailored towards cryptographic applications, has been applied to the formalization of security proofs [23]. Early versions of EasyCrypt integrated with Rocq through the CertiCrypt library [24]. However, this integration was later discontinued, and EasyCrypt is no longer foundational, as it relies on the results returned by unverified SMT solvers.

3.5.3 Stochastic Processes

Formalizations of *stochastic processes* in ITPs are closely related to our work. Stochastic processes are mathematical objects that represent collections of random variables indexed by time. Prominent examples of stochastic processes include martingales, Markov chains and MDPs.

Martingales A stochastic process is called a *martingale* if the expected value of the next observation is equal to the current one. Martingales play an important role in probability theory and are frequently used to model gambling scenarios, stock prices, and other applications in finance.

Several formal libraries for martingales have been developed in proof assistants. In PVS, Daumas et al. [46, 47] formalized a restricted version of the *Doob-Kolmogorov inequality* and applied it to the analysis of rounding errors in floating-point computations. In Rocq, martingale formalizations cover *Dvoretzky's stochastic approximation theorem* and the *martingale convergence theorem* [156]. The Lean the library on measure theory has been used to formalize stochastic processes and martingales, including *Doob's martingale convergence theorem* [163]. These results were later replicated and generalized in Isabelle/HOL [100]. Towards applications in *finance*, Jaeger formalized filtrations, stopping times, and arbitrage theory in Mizar [92]. Similarly, in Isabelle/HOL, results from probability theory have also been applied to study the pricing of derivatives [53, 54].

Markov Chains & Markov Decision Processes Formal libraries for Markov chains were developed in HOL4 [89], Lean [114], and in Isabelle/HOL by Hölzl [83]. The Isabelle/HOL library was used in the formal analysis of distributed protocols [86], the running time of probabilistic pGCL programs [82], and the verification of probabilistic model checking [87]. More broadly, a general framework for formalizing a variety of probabilistic system types underlying stochastic processes has been developed in Isabelle/HOL [85].

The formalization of Markov chains in Isabelle/HOL was later extended to MDPs [83]. This work includes an analysis of reachability properties and the development of a certificate checker for reachability probabilities based on fixed point certificates (see Section 3.3). In addition, it defines an operational semantics for pGCL in terms of MDPs and formally proves its equivalence to the denotational semantics (see Section 3.5.2).

Classical iterative dynamic programming algorithms for MDPs have also been formalized in Rocq by Vajjha et al. [155] and in Isabelle/HOL by Chevallier and Fleuriot [39]. These probabilistic planning algorithms compute policies for MDPs that maximize the *expected total discounted reward*, i.e. the sum of all expected rewards, where earlier rewards are weighted more heavily than later ones. While both *value iteration* and *policy iteration* algorithms have been verified to optimize expected discounted rewards, prior work did not refine these abstract algorithms into efficiently executable implementations. In Section 4.1.1, we address this gap by providing practical, verified solvers for these algorithms.

In the domain of probabilistic model checking, the *interval iteration* algorithm is a sound and efficient variant of value iteration [64]. Before applying this algorithm, the MDP must be transformed into a canonical form to ensure convergence and performance. As a step towards a fully verified probabilistic model checker, verified versions of the *strongly connected component* and *maximal end component decomposition* algorithms have been developed [75, 76]. These algorithms have been refined into low-level imperative LLVM code with competitive performance, making them suitable as drop-in replacements for unverified implementations.

4 Formally Verified Algorithms for Markov Decision Processes

The main contribution of this thesis is the formalization of a library of executable algorithms for MDPs in Isabelle/HOL. Our work spans a range of methodologies, application areas, and implementation architectures.

First, we address both probabilistic planning and model checking for MDPs, focusing on reachability objectives and expected total reward properties. Second, our verified implementations explore not only exact arithmetic, but also floating-point arithmetic with directed rounding techniques to ensure soundness. We further contrast different approaches to code generation: refinement to imperative code using the *Isabelle Refinement Framework*, and the export of functional code via Isabelle's built-in code generation facilities. In the latter, we adopt a *locale-based approach* to develop reusable data structures.

To enhance the trustworthiness of MDP solutions, our work applies multiple verification methodologies. We provide fully verified algorithm implementations as well as certification techniques that validate the output of unverified solvers. Across all contributions, we prioritize practical algorithms and efficient execution. In the following, we outline these contributions as presented in our publications.

4.1 MDP Planning

Our first line of work contributes verified implementations of algorithms for probabilistic planning. These projects are presented in *Core Publication 1* [145] and *Core Publication 2* [144]. We begin by focusing on classical *dynamic programming algorithms* for explicitly represented MDPs in Section 4.1.1. We then extend our work in Section 4.1.2 to a verified approximate solution method that exploits structure commonly found in large MDPs.

In probabilistic planning [134], the goal is to make decisions under uncertainty. Given an MDP that models the environment, along with *rewards* awarded for each action, the objective is to optimize the agent's action choices to maximize long-term cumulative rewards. A standard illustrative example are *grid world* problems, as shown in Figure 4.1. In such scenarios, a robot must navigate a grid to reach a goal state while avoiding obstacles. The robot can move in the four cardinal directions, but each movement may fail due to environmental uncertainty, causing the robot to slip and move in an

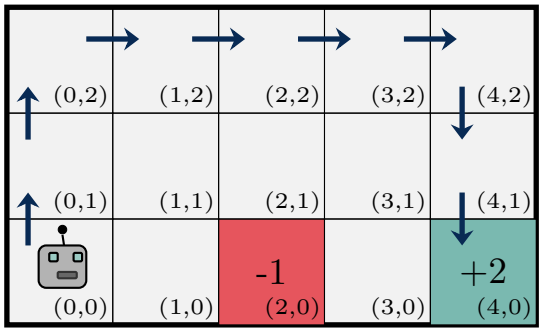


Figure 4.1: A grid world problem: the robot must navigate a grid to reach the goal state (bottom right) while avoiding obstacles (bottom middle). Initially located in the bottom left corner, the robot can move in the four cardinal directions, though actions may fail occasionally. A positive reward is given for reaching the goal state, while hitting an obstacle incurs a penalty. The robot may choose between shorter, riskier paths or longer, safer routes. Parts of an exemplary conservative policy are shown as arrows in the figure.

unintended direction. Reaching the goal state yields a positive reward, while colliding with an obstacle incurs a cost, expressed as a negative reward. Our grid world problem can be readily modeled as an MDP with 15 states, where the robot's state is represented by its position on the grid, and the actions correspond to the four possible movements.

Expected Total Rewards In Isabelle/HOL, we formalize two fundamental settings in probabilistic planning: the finite-horizon and infinite-horizon scenarios. In the *infinite-horizon* setting, the objective is to find a policy that maximizes the expected sum of rewards accumulated over an infinite sequence of discrete time steps. The quantity being optimized is known as the *expected total discounted reward*, denoted by $\mathbb{E}_\gamma^{\max}$. To ensure convergence and encourage short-term gains over delayed ones, rewards are discounted by a factor $\gamma \in [0, 1)$ at each time step. This setting is well understood, and several classical solution methods are available to compute optimal policies—most notably value iteration, policy iteration, and a transformation to linear programming [134]. The resulting policies are simple, as optimal policies are *memoryless and deterministic*: They depend solely on the current state, not on the history of prior states or actions.

In the *finite-horizon setting*, the goal is to find a policy that maximizes the expected sum of rewards obtained in a fixed, finite number of time steps; called the *expected total reward* for finite-horizon problems. The standard solution technique is *backward induction*, a dynamic programming algorithm. The method proceeds by starting at the final time step, computes the optimal policy for each state, then iteratively steps backward in time to determine the full policy. Unlike the infinite-horizon case, the optimal policies in finite-horizon problems are not necessarily memoryless: Action selection may depend on the number of steps remaining.

4.1.1 Planning with Tabular Solution Methods

As part of *Core Publication 1*, we present a formalization of MDPs with rewards and stochastic action choice in Isabelle/HOL.[1] This work extends the author's master's thesis [143], which included a formalization of value iteration and policy iteration. However, that thesis did not provide efficiently executable implementations. As part of this dissertation, we extend and restructure the MDP library, design a methodology for deriving verified implementations, and expand the suite of algorithms to include *backward induction*, *Gauss-Seidel value iteration*, and *modified policy iteration*.

MDPs with Rewards We define MDPs in Isabelle/HOL as a *locale*, parameterized by types for states and actions, and fixing a *transition kernel* P that maps state-action pairs to probability distributions over successor states (see Figure 2.2). Additionally, for each state, we define a set of *enabled actions*. In this context, we extend that definition of MDPs with a *reward function* $\mathrm{rew} \colon S \times \mathit{Act} \to \mathbb{R}$, which assigns a real-valued reward to each state-action pair.

Our formal analysis of probabilistic planning builds on classical results [134], where the *Bellman operator* plays a central role. The operator expresses the expected value of the next state we observe, given a current value estimate $v : S \to \mathbb{R}$, the current state $s \in S$ and a policy $\sigma : S \to \mathit{Act}$. For simple *memoryless deterministic* (MD) policies, the Bellman operator $\mathcal{E}_\gamma^\sigma : (S \to \mathbb{R}) \to (S \to \mathbb{R})$ is defined as:

$$\mathcal{E}_\gamma^\sigma(v)(s) = \mathrm{rew}(s, \sigma(s)) + \gamma \cdot \sum_{s' \in S} P(s, \sigma(s), s') \cdot v(s').$$

We formally prove that the unique fixed point of $\mathcal{E}_\gamma^\sigma$ equals $\mathbb{E}_\gamma^\sigma$, the expected total discounted reward of the policy. Furthermore, let Π_{MD} denote the set of MD policies. Then the unique fixed point of the *Bellman optimality operator* $\mathcal{E}_\gamma^{\max}(v) \coloneqq \max_{\sigma \in \Pi_{\mathrm{MD}}} \mathcal{E}_\gamma^\sigma(v)$ is the maximum possible reward $\mathbb{E}_\gamma^{\max}$. Since the operator is a contraction mapping, repeated application to any initial value estimate yields a sequence that converges to $\mathbb{E}_\gamma^{\max}$. In our formal proofs, we follow the structure of the textbook exposition by Puterman [134].

Verified Dynamic Programming Algorithms *Value iteration* is a standard algorithm for computing the maximum value $\mathbb{E}_\gamma^{\max}$ of an MDP. It directly implements a fixed point iteration of the Bellman optimality operator $\mathcal{E}_\gamma^{\max}$, which converges to $\mathbb{E}_\gamma^{\max}$. The iteration continues until the maximum difference between two consecutive value estimates at any state falls below an error bound derived from the error threshold $\epsilon > 0$. At that point, we return a policy that achieves the maximum in the definition of $\mathcal{E}_\gamma^{\max}$. We formally prove that this policy is ϵ-optimal, i.e. its value differs from $\mathbb{E}_\gamma^{\max}$ by at most ϵ at every state.

We also formalize *Gauss-Seidel value iteration*, a variant of value iteration that updates value estimates in-place. This in-place update often speeds up convergence. Our formal

[1] Our work is available online: `https://github.com/schaeffm/mdps-isabelle-hol`

convergence proof of the algorithm closes a gap in the textbook argument [145, Thm. 3]: the proof by Puterman [134] assumes, without justification, that the maximum in the Gauss-Seidel variant of $\mathcal{E}_\gamma^{\max}$ is attained by a single $\sigma \in \Pi_{\text{MD}}$ across all states. We formally construct such a policy and validate the proof in Isabelle/HOL.

Policy iteration follows a different approach: rather than estimating $\mathbb{E}_\gamma^{\max}$ directly, it successively evaluates and improves a candidate policy. For the evaluation phase, we compute the value function by solving a system of linear equations, using a verified implementation of the Gauss-Jordan algorithm [151]. Next, we improve the policy based on $\mathcal{E}_\gamma^{\max}$, similar to value iteration. This process continues until convergence to an optimal policy.

Finally, *modified policy iteration* is a hybrid approach that avoids exact policy evaluation. Instead, it approximates the value of a policy by applying the Bellman operator a fixed number of times between policy improvements.

We extract executable code from our formalizations in Isabelle/HOL for all of the algorithms described above, as well as the backward induction procedure for finite-horizon problems. These implementations rely on exact arithmetic: probabilities and rewards are represented using rational numbers, implemented as fractions of integers. For data structures such as sets, mappings, and MDP representations, we adopt a *locale-based approach*. We define locales to specify abstract interfaces of data structures and instantiate them with concrete implementations. In Isabelle/HOL, this design allows multiple different implementations to coexist for the same abstract type.

Experimental Evaluation We experimentally evaluate the scalability of our verified algorithms using benchmarks from the probabilistic track of the *IPC 2018* planning competition. The experiments show that our verified implementations using exact arithmetic scale to problems with up to hundreds of thousands of states.

Among the verified algorithms, standard value iteration proves to be the most efficient in practice. Although the Gauss-Seidel variant converges faster in theory, the in-place updates lead to larger fractions in the value vectors, which negatively impacts performance. Policy iteration delivers exact solutions but scales less favorably, as it requires solving a system of linear equations in each iteration. This becomes a computationally expensive task for larger systems.

Based on these findings, we implement a certification-based approach to improve practical performance. Since value iteration converges from every initial value estimate, the algorithm can be *hot-started* using approximate solutions from unverified solvers. We therefore propose to compute an approximate value with a floating-point implementation, then use our verified implementation to refine and certify the result. If the floating-point solution is inaccurate due to rounding errors, additional iterations are performed to meet the desired error threshold. Our verified certificate checker can also be used to validate exact value vectors.

MDP Library by Hölzl In Isabelle/HOL, an existing formalization of MDPs has been developed by Hölzl [83]. However, due to technical considerations, our definition of

```
1  locale MDP =
2    fixes   Act :: 's ⇒ 'a set
3      and   P :: 's × 'a ⇒ 's pmf
4    assumes
5      Act_ne: ⋀s. Act s ≠ ∅
```

```
1  locale MDP =
2    fixes K :: 's ⇒ 's pmf set
3    assumes
4      K_wf: ⋀s. K s ≠ ∅
```

Figure 4.2: Comparison of the two definitions of MDPs in Isabelle/HOL. We contrast our definition of MDPs with labelled actions on the *left* with Hölzl's definition of MDPs with unlabelled actions on the *right*. For the latter, the transition distributions and action sets are unified in a single function K, the transition kernel.

MDPs with rewards is not based directly on this prior work. Our formalization differs from that of Hölzl in several important ways. A comparison of the two definitions is shown in Figure 4.2.

First, our definition supports uncountable state spaces and stochastic action choice, making it more general. This added expressiveness allows us to follow standard textbook presentations of MDP algorithms, which often rely on the concept of randomized policies—policies that select actions probabilistically. Such policies sometimes simplify the proofs of properties like the optimality of memoryless policies [134].

Hölzl models the *transition kernel*—the transition probabilities—as a function from states to sets of probability distributions over states; effectively actions are unlabelled. In contrast, we model the action sets explicitly: the transition kernel maps state-action pairs to probability distributions. This design choice simplifies implementation tasks, such as parsing system models with labelled actions. It also avoids special handling for semantically distinct actions that have the same transition probabilities but yield different rewards.

On the other hand, the original formalization offers advantages in terms of simplicity. Encoding all behavior into a single object can lead to more concise proofs with fewer indirections. Moreover, Hölzl does not use policies, but rather *configurations* to describe the action selection. A configuration combines the MDP and the policy in a single entity that can be interpreted directly as a Markov chain. It remains future work to merge these two approaches into a unified and general formalization of MDPs in Isabelle/HOL.

4.1.2 Approximate Planning for Factored MDPs

If we attempt to solve large-scale MDPs using the algorithms described in Section 4.1.1, we quickly encounter the *curse of dimensionality*: for many systems, the number of states grows exponentially with respect to the size of a compact description of the MDP. To illustrate this, consider again the grid world example from Figure 4.1. For a grid of size $m \times n$, the state space consists of $m \cdot n$ states. Yet the environment itself can be concisely described using only two integers to specify the grid size, and two pairs of integers to identify the positions of the obstacle and goal states.

Scaling formally verified MDP algorithms to large systems can be approached in ways beyond representing the state space more compactly. In this thesis, we employ three orthogonal strategies: 1) we develop verified implementations that use efficient floating-point arithmetic (see Section 4.2.1), 2) we certify or validate results produced by highly optimized, unverified implementations (see Section 4.2.2), and 3) we exploit structural regularities present in large-scale MDPs.

In the following, we focus on the latter two strategies, and present a verified implementation of *approximate policy iteration* (API) for *factored MDPs* [63], i.e. MDPs in which the state space, transition function and reward function are described with a compact, structured representation. This work is detailed in *Core Publication 2* and the accompanying Isabelle/HOL formalization.[2]

Factored MDPs As a first step, we formalize the model of factored MDPs, as introduced by Guestrin et al. [63], in Isabelle/HOL. Factored MDPs offer a compact representation of MDP by structuring both the state space and the transition and reward functions. In practice, a variety of modeling languages support such representations, including PRISM [108], JANI [36], and PPDDL [164]. A potential extension of our work is to define formal semantics for these languages, which could serve as the foundation for fully verified solvers. On the implementation side, large MDPs are often represented using *decision diagrams*, which enable efficient computation [79].

In the model proposed by Guestrin et al., the state space is defined as a product of individual state variables, each of which can take on a finite number of integral values. For example, in the grid world scenario, the state space can be modeled as a product of the robot's possible x and y coordinates.

The transition and reward functions are constructed from a combination of *scoped functions*. Each scoped function only observes a small subset of the state variables, which allows for a smaller description. To further compress the description, each MDP includes a *default action*, which captures the natural evolution of the system when no specific action is taken. For instance, in a variant of the grid world, the robot's future x coordinate might depend only on its current x value. Therefore it is unnecessary to consider the y coordinate for that part of the model.

In Isabelle/HOL, we implement a library for scoped functions and use it to define the transition and reward structure of factored MDPs. We then connect our definition of factored MDPs to the existing formalization of tabular MDPs introduced in Section 4.1.1. This connection allows us to reuse our analysis of expected total discounted rewards in the factored MDP setting. Additionally, we increase confidence in our definitions by relating them to an existing verified representation of MDPs.

Approximate Policy Iteration Building on our library of factored MDPs in Isabelle/HOL, we define the *approximate policy iteration* (API) algorithm for factored MDPs. API is an approximate variant of policy iteration that incrementally improves a candidate policy. Each iteration consists of three phases: approximate policy evaluation

[2] Our work is available online: `https://github.com/schaeffm/fmdp_isabelle`

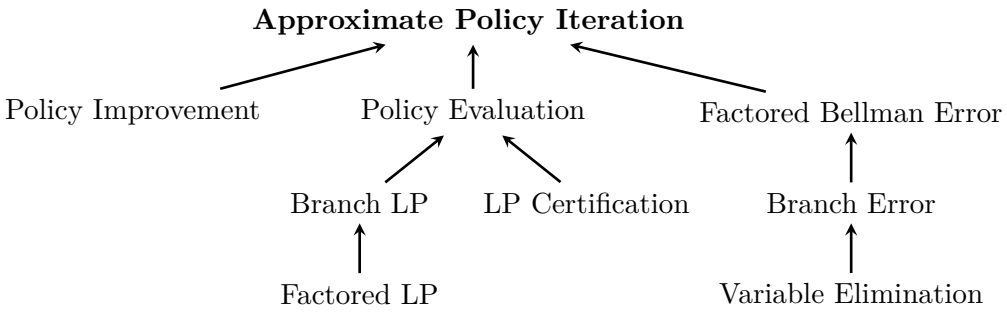


Figure 4.3: Overview of the API algorithm and its subprocedures. The term *branch* refers to the individual components of a decision list policy.

using linear programming, policy improvement, and computation of the Bellman error. The Bellman error is defined as the maximum difference between a value estimate v and the result of applying the Bellman optimality operator $\mathcal{E}_\gamma^{\max}(v)$ across all states. The algorithm terminates when the Bellman error falls below a predefined threshold or the policy converges. While API does not offer *a priori* optimality guarantees, we can derive suboptimality bounds upon convergence.

During the *policy evaluation* phase, the value of the current policy is approximated using a predefined set of *basis functions*. These basis functions are domain-specific and are designed to strike a balance between expressiveness and computational efficiency. On one hand, their linear combination should closely approximate the true value function, on the other, each basis function should have a small scope, i.e. depend only on a limited subset of state variables. The value function is expressed as a linear combination of these basis functions. *Linear programming* is used to optimize the associated weights and minimize the approximation error.

Policies in API are represented as *decision lists*, which are ordered lists of state patterns paired with corresponding actions. The agent selects the first pattern in the list that matches the current state and executes the associated action. In the *policy improvement* phase, the decision list is updated based on the newly computed weights.

In the final phase, we compute the *Bellman error* of the current policy, which measures the difference between the current value function and the result of applying the Bellman operator to it. This error serves as a stopping criterion: once it drops below a specified threshold, the algorithm halts. We formally verify a *variable elimination* procedure that computes the Bellman error efficiently, without an explicit enumeration of the entire state space.

Methodology API relies on a complex hierarchy of subprocedures, including variable elimination, linear program construction and solving, and policy improvement. We implement the algorithm in a modular fashion using Isabelle/HOL locales to structure the development. An overview of the algorithm and its subcomponents is shown in Figure 4.3.

Each subprocedure is implemented as a locale that assumes the existence of its dependencies and their adherence to a specification. This modular structure enables us to reason about correctness locally within each component and prepares the subprocedures for reuse in other contexts. We use the hierarchy of locales consistently in both the abstract proofs and during the code generation phase.

To solve linear programs (LPs) in a trustworthy manner, we adopt a certificate-checking approach in our verified implementation. While we rely on exact but unverified solvers to compute LP solutions, these results are validated using a formally verified checker. The certification is based on the duality theorem of linear programming, which implies that an optimal dual solution can certify the optimality of the corresponding primal solution. We implement a general-purpose, verified LP certificate checker in Isabelle/HOL that is not specific to MDPs and can be reused across verification projects. Currently, we support the exact LP solvers SoPlex [35] and QSopt_ex [10] as backends.

We evaluate our implementation on computer network models used in the original presentation of the algorithm [63]. Our results show that verified API with exact arithmetic scales to systems with up to 10^{12} states, several orders of magnitude larger than the 10^5 states handled by the verified tabular algorithms discussed in Section 4.1.1. The primary bottleneck lies in the exact LP solvers, which struggle to solve the linear programs generated from even larger instances [144, Table 1].

4.2 Probabilistic Model Checking

In artificial intelligence applications of MDPs—such as planning and reinforcement learning—MDPs serve as models of the environment and the behavior of an agent. The objective is to identify a policy that optimizes a specific goal, such as maximizing rewards or minimizing costs. Another key application domain for MDPs is the verification of probabilistic systems, specifically through *probabilistic model checking* (PMC). In this setting, MDPs are used to model the behavior of systems, protocols, or probabilistic programs, with the goal of verifying system properties. Compared to planning, in PMC we often require a subtly different interpretation of the MDP: actions represent non-deterministic branching within the system, while transition probabilities capture stochastic behavior, such as potential failures.

PMC has been applied to the analysis of wireless communication protocols [98], network-on-chip architectures [139], and models of system reliability and performance [16]. In these applications, PMC is used to answer questions regarding the system's performance, reliability, and safety. These questions are typically answered by probabilistic model checkers, which reduce the problem to computing an *expected total reward* or the *reachability probabilities* of certain sets of states. However, since model checkers are complex, human-engineered software systems, the correctness of their results cannot be taken for granted. Reported sources of errors include: (i) implementation bugs, (ii) unsound algorithms [64], (iii) numerical inaccuracies due to floating point arithmetic [70], and (iv) errors in third-party libraries such as LP solvers [72]. While partial mitigation

strategies exist—such as extensive testing, the use of exact arithmetic, or certificate checkers for LP solutions (see Section 4.1.2)—these issues remain critical.

In this line of work, based on *Core Publication 3* [103] and *Core Publication 4* [38], we present two complementary contributions to the verification of PMC, both designed to address the aforementioned issues. First, we provide a formally verified floating-point implementation of the *interval iteration* algorithm for computing reachability probabilities in MDPs [64]. Second, we introduce an efficient certification approach for verifying the results of probabilistic model checkers using *fixed point certificates*. We formally prove the soundness of these certificates in Isabelle/HOL and develop a formally verified checker for the certificates that can be used to validate the results of unverified model checkers.

These two techniques, certificate checkers and verified implementations, offer distinct strengths. Certificate checkers are generally simpler to implement and verify, and they can be used by independent third parties to validate results. Verified implementations, on the other hand, provide stronger correctness guarantees and avoid the potentially costly step of generating certificates.

4.2.1 Verified Interval Iteration with Floating-Point Arithmetic

The value iteration algorithm is a popular and efficient solution method, not only in the context of probabilistic planning (Section 4.1.1), but also in probabilistic model checking, where it is used to compute reachability probabilities. However, the standard stopping criteria used in value iteration have been shown to be insufficient for PMC applications [64]. Specifically, even when the value estimates seem to stabilize and change by less than small threshold, the results may still deviate significantly from the true value.

The *interval iteration* algorithm [64] was proposed to address this soundness issue of value iteration when computing reachability probabilities in MDPs. It can be applied to compute both maximum and minimum reachability probabilities to a given target set T. For the maximum case, $\mathbb{P}^{\max}(\Diamond T)$ represents the probability of reaching T under a strategy that maximizes the chance of doing so. Conversely, $\mathbb{P}^{\min}(\Diamond T)$ denotes the probabilities when the agent actively tries to avoid the target set.

In this section, based on *Core Publication 3* [103], we present our formally verified floating-point implementation of the interval iteration algorithm.[3] The key idea of interval iteration is to execute two instances of value iteration in parallel. One starts with an initial value of zero for all non-target states and produces a sequence of *lower bounds*. The other instance starts from one, computing *upper bounds*. At every iteration, the true reachability probability is contained between these two bounds. This ensures that the error is at most the size of the interval, and the stopping criterion becomes trivial: we terminate once the interval is smaller than a given tolerance.

Importantly, interval iteration can be implemented with floating-point arithmetic, retaining formal guarantees on the results. Even in the presence of numerical error,

[3] Our work is available online: `https://doi.org/10.4121/bf0fef24-4f0f-4de6-a58d-07b9ba601804`

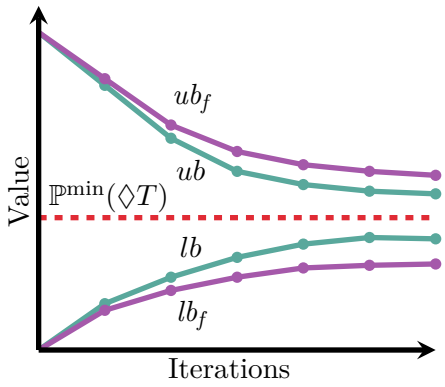


Figure 4.4: Illustration of a typical run of the interval iteration algorithm, focusing on a single state. The dashed line represents the true reachability probability. The solid inner lines (ub, lb) show the iterates of exact interval iteration. The solid outer lines (ub_f, lb_f) denote the corresponding floating-point iterates using directed rounding. Adapted from Kohlen et al. [103, Fig. 4].

sound bounds can be obtained by applying *directed rounding*, i.e. always rounding down for lower bounds and rounding up for upper bounds [70]. We illustrate the behavior of interval iteration on a representative example in Figure 4.4.

Isabelle/HOL Formalization We define interval iteration in Isabelle/HOL and formally prove its soundness and convergence. Our development builds on the formal library of MDPs by Hölzl [83], which includes an analysis of reachability properties. In this section, we focus on computing maximum reachability probabilities, though the similar case for minimum probabilities is also supported in our formalization. Both value iteration instances in interval iteration—the lower and upper bound computations—repeatedly apply the Bellman operator $\mathcal{B}^{\max}\colon (S \to [0,1]) \to (S \to [0,1])$ to an initial upper and lower bound:

$$\mathcal{B}^{\max}(v)(s) \;=\; \begin{cases} 1 & \text{if } s \in T, \\ \max\limits_{a \in Act(s)} \sum\limits_{s' \in S} P(s,a,s') \cdot v(s') & \text{if } s \in S \setminus T. \end{cases}$$

Unlike the discounted Bellman operator $\mathcal{E}_\gamma^{\max}$, the operator $\mathcal{B}^{\max}$ does not necessarily have a *unique* fixed point. As a result, the fixed point iteration may converge to a *spurious* fixed point that is not the least fixed point $\mathbb{P}^{\max}(\Diamond T)$ of the Bellman operator. This issue arises due to the presence of *maximal end components* (MECs) in the MDP: maximal subsystems that the process never needs to leave once entered. Within certain MECs, states can mutually support value estimates that are too high.

To address this, interval iteration operates on *reduced* MDPs, in which problematic MECs have been removed. The reduction process involves collapsing each MEC into

a single state and eliminating specific classes of MECs depending on the reachability probabilities being computed.

We formalize these reductions in Isabelle/HOL for both minimum and maximum reachability probabilities. Our proofs establish that the preprocessing is sound, i.e. the reachability probabilities in the reduced MDP are identical to those in the original. Moreover, we refine the existing pen-and-paper proofs [64]: in the original proof, the authors vaguely state that the reduced and original versions have corresponding policies with equal reachability probabilities, while we reason rigorously via finite-horizon reachability probabilities.

For reduced MDPs, we prove that interval iteration converges to the correct fixed point. The convergence rate depends on the smallest probability in the MDP. We follow the proof strategy by Haddad and Monmege [64] and show that in a reduced MDP, every action has a non-zero chance of moving the system either towards the target or towards a sink state from which the target is unreachable.

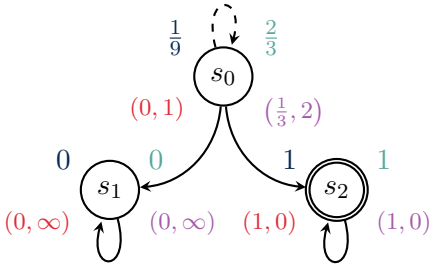
Efficient Implementation In *Core Publication 3*, we refine the abstract interval iteration algorithm into an efficient floating-point implementation using the *Isabelle Refinement Framework* (IRF). The IRF structures and partially automates the process of generating efficient code from formalized algorithms in Isabelle/HOL [110].

As a first step, we extend the IRF to support proving bounds on floating-point computations with directed rounding. Our goal is to compute floating-point upper bounds for the upper-bounding instance and lower bounds for the lower-bounding instance of the interval iteration algorithm. This allows us to conclude that the true reachability probabilities lie within the interval formed by these two floating-point bounds. Such guarantees are achieved by using directed rounding modes: rounding down when computing lower bounds and rounding up for upper bounds.

Based on this extension, we generate efficient LLVM code from the formalization. The resulting implementation performs floating-point computations using directed rounding, while preserving the formal guarantee that the true reachability probabilities are contained in the computed interval. However, since floating-point arithmetic introduces approximation, convergence to the true reachability probabilities cannot be formally proven directly with this approach and must instead be established experimentally.

Our experimental results show that the verified implementation performs competitively with the unverified implementation of the same algorithm in mcsta [71], evaluated on benchmark models from the *Quantitative Verification Benchmark Set* [74]. In all instances, the target error bound of 10^{-6} was successfully achieved using directed rounding. While this technique may increase the number of required iterations in some cases, the overall impact on performance is negligible [103, Fig. 5].

Currently, MDP reduction is not part of the verified implementation, as the soundness of the interval iteration algorithm is preserved even on unreduced MDPs. Proving formal convergence guarantees in the floating-point setting remains future work. This would require two main extensions: extending the IRF with facilities for reasoning about floating-point rounding errors, and integrating the MDP reduction directly into the ver-



$\min\left\{\frac{1}{9}, \frac{1}{2}\cdot 0 + \frac{1}{2}\cdot 1\right\} \overset{\checkmark}{\leq} \frac{1}{9}$

$\max\left\{\frac{2}{3}, \frac{1}{2}\cdot 0 + \frac{1}{2}\cdot 1\right\} \overset{\checkmark}{\leq} \frac{2}{3}$

$\min\left\{0, \frac{1}{2}\cdot 0 + \frac{1}{2}\cdot 1\right\} \overset{\checkmark}{\geq} 0$

$1 + \max\left\{0, \min\left\{\infty, 0\right\}\right\} \overset{\checkmark}{\leq} 1$

$\max\left\{\frac{1}{3}, \frac{1}{2}\cdot 0 + \frac{1}{2}\cdot 1\right\} \overset{\checkmark}{\geq} \frac{1}{3}$

$1 + \min\left\{2, \min\left\{\infty, 0\right\}\right\} \overset{\checkmark}{\leq} 2$

Figure 4.5: An MDP with three states and two actions (solid and dashed edges, respectively), uniform probabilities, and a target set $T = \{s_2\}$. The annotations next to the states are certificates for upper bounds on minimum (top left), upper bounds on maximum (top right), lower bounds on minimum (bottom left), and lower bounds on maximum (bottom right) reachability probabilities. On the right, we demonstrate the verification of the certificates (for s_0 only; similar inequalities have to be checked for the other states as well). Since all certificates are valid we may deduce that $\frac{1}{3} \leq \mathbb{P}^{\max}_{s_0}(\Diamond T) \leq \frac{2}{3}$, the exact value is $\mathbb{P}^{\max}_{s_0}(\Diamond T) = \frac{1}{2}$.

ified implementation. For the latter, an efficient and formally verified implementation of MEC decomposition is already available in Isabelle/HOL [75, 76].

4.2.2 Fixed-Point Certificates

In *Core Publication 4* [38], we introduce a certificate generation and checking approach for probabilistic model checkers.[4] Our certificates for MDP model checking are both simple and efficient to validate using a formally verified checker. Validation relies only on *state-local* checks, so we require no graph-theoretic reasoning during the verification phase. In addition to being straightforward to check, the certificates are also easy to generate in model checkers, with only modest overhead. All certificate formats are both *sound* and *complete*: they provide valid certificates for the corresponding properties, and the exact value of the property can be certified. The certificates cover both minimum and maximum reachability probabilities, as well as minimum and maximum expected total rewards. For reachability analysis, we support certificates for both *qualitative* and *quantitative* properties.

To illustrate the core challenges in designing sound and complete certificates, we use a simple MDP example shown in Figure 4.5. The goal is to reach the target set $T = \{s_2\}$. In this case, the minimum reachability probabilities are $\mathbb{P}^{\min}(\Diamond T) = (0_{s_0}, 0_{s_1}, 1_{s_2})$ using the dashed action at s_0, whereas the vector of maximum reachability probabilities is $\mathbb{P}^{\max}(\Diamond T) = (\frac{1}{2}_{s_0}, 0_{s_1}, 1_{s_2})$.

[4] Our work is available online: `https://zenodo.org/records/14626586`

Certificates for Upper Bounds The optimal reachability probabilities $\mathbb{P}^{\text{opt}}(\Diamond T)$ for $\text{opt} \in \{\min, \max\}$ can be characterized as the least fixed point of the Bellman operator $\mathcal{B}^{\text{opt}}$, defined in Section 4.2.1. The Bellman operator is *monotonic* on the complete lattice $(S \to [0,1])$ with the point-wise order $\leq$. By elementary fixed point theory, $\mathcal{B}^{\text{opt}}$ has the least fixed point

$$\text{lfp}\, \mathcal{B}^{\text{opt}} = \inf\{v : S \to [0,1] \mid \mathcal{B}^{\text{opt}}(v) \leq v\} = \mathbb{P}^{\text{opt}}(\Diamond T).$$

Certificates for upper bounds require no additional information. If a value vector v satisfies $\mathcal{B}^{\text{opt}}(v) \leq v$, it directly follows that v is an upper bound on the true reachability probabilities: $\mathbb{P}^{\text{opt}}(\Diamond T) \leq v$.

Certificates for Lower Bounds Certifying lower bounds is more complex than certifying upper bounds. The condition $\mathcal{B}^{\text{opt}}(v) \geq v$ alone is not sufficient to conclude that $\mathbb{P}^{\text{opt}}(\Diamond T) \geq v$, because the Bellman operator may have multiple fixed points. For example, consider the case of $\mathbb{P}^{\min}(\Diamond T)$ in Figure 4.5: the value vector $v = (1_{s_0}, 1_{s_1}, 1_{s_2})$ satisfies $\mathcal{B}^{\min}(v) \geq v$, yet it is clearly not a valid lower bound.

To rule out such cases, we must ensure that states capable of avoiding the target set indefinitely are assigned zero probability in the value vector. To this end, we employ a *ranking function* $r \colon S \to \overline{\mathbb{N}}$. Intuitively, the ranking function encodes a distance to the target states under a policy that attempts to avoid the target for as long as possible.

We verify the correctness of the ranking function through a check analogous to the one used for upper bounds on reachability: it must be an upper bound on the true distances. Additionally, we require that only states with finite rank may be assigned a non-zero probability in the value vector. The formal definition of the ranking function and the associated checks are provided in the original publication [38].

Certificates for lower bounds on $\mathbb{P}^{\max}(\Diamond T)$ introduce an additional complication. Consider the value function $(1_{s_0}, 0_{s_1}, 1_{s_2})$ with ranking $(1_{s_0}, \infty_{s_1}, 0_{s_2})$. Although they pass the checks introduced above, the value function is not a valid lower bound for $\mathbb{P}^{\max}(\Diamond T)$. The issue is that state s_0 can support its own value by looping via the dashed action. However, to actually reach the target, it must eventually take the solid action, which causes a loss in probability mass. To address this, we refine the distance to the target set: it is computed using only actions that *do not decrease the probability of reaching the target*. In the example, this excludes the solid action. As a result, the distance from s_0 to the target becomes infinite, invalidating the certificate.

Certificates for Rewards We also provide and formally verify certificates for *expected total reward* problems. These build upon certificates for *qualitative reachability*, where the goal is to determine whether a target state is reachable with probability one (almost surely) or with non-zero probability. A key contribution is the certificate for *non-almost sure* reachability. The corresponding soundness and completeness proofs are non-trivial and require subtle reasoning [38, Proposition 2].

Isabelle/HOL Formalization All certificates presented in *Core Publication 4* are sound and complete. We formalize the fixed point certificates in Isabelle/HOL. While we formally prove soundness for all certificate formats, completeness for the *non-almost sure* reachability certificates is left as future work. However, this property is also not required for validating certificates with a formally verified checker.

Our verified certificate checker builds upon the formalization of MDPs by Hölzl, who had previously verified similar certificates for upper bounds on maximal reachability and lower bounds on minimal probabilities [83]. From our formalization, we generate executable code that can be used to automatically check certificates for reachability and reward properties.

Experimental Evaluation We extend the probabilistic model checker Storm [79] with support for generating our fixed point certificates. In doing so, we ensure that the computed reachability probabilities are produced in a way that passes the formal certificate checks. For example, lower-bound certificates require that applying the Bellman operator increases the value vector at every state.

There are multiple approaches for computing such value vectors. Exact algorithms using rational arithmetic, such as policy iteration or linear programming, provide strong guarantees but incur higher computational cost. Alternatively, more efficient approximate algorithms like interval iteration with directed rounding can be used (see Section 4.2.1). However, these algorithms may not always yield valid certificates: for instance, not every upper bound is a post fixed point of the Bellman operator. The ranking functions required for lower-bound certificates can be computed using value iteration.

We evaluate the performance of certificate generation and checking on a broad set of benchmarks, including the *Quantitative Verification Benchmark Set* (QVBS) [74]. Our experiments show that certificate generation using policy iteration typically incurs an overhead of less than a factor of two. The verified certificate checker is able to validate certificates for systems with up to 10^6 states in under 30 s.

When scaling to larger systems, the main bottleneck is the parsing of certificates into the verified checker. Currently, certificates are stored in plain-text format. As future work, we plan to adopt a compact binary format to reduce both certificate size and parsing time.

5 Conclusion

In this thesis, we have presented a suite of formally verified algorithms for MDPs. These algorithms span a broad spectrum of MDP-related problems, including both finite horizon and infinite horizon scenarios in probabilistic planning, as well as the computation of reachability probabilities and expected rewards in probabilistic model checking. Our contributions are diverse not only in terms of the algorithms we verified but also in the verification techniques employed and the classes of errors we targeted. The core methodology relies on the Isabelle/HOL proof assistant and its code generation capabilities. Throughout our work, we explored and contrasted numerous verification techniques—such as certificate checking, directed rounding and exact arithmetic—alongside refinement approaches and the generation of imperative or functional code.

Our results highlight Isabelle/HOL as a highly suited environment for the verification of MDP algorithms. Key enablers include the libraries available in the AFP, powerful automated proof tools and features such as locales, which support modular development of large-scale formalizations. Our findings demonstrate that the mechanization of algorithms not only deepens our understanding of their properties but also yields correctness proofs that are more precise. Additionally, using ITPs early on during the design of new algorithms, as we did in this thesis, leads to a more efficient and less error-prone process.

We addressed the four sources of errors in MDP software identified in the introduction. *Algorithmic errors*, such as the unsound stopping criterion of value iteration, can be eliminated by formally verifying the algorithms themselves. To mitigate *implementation errors*, we verified executable versions of our algorithms. Additionally, *third-party libraries* are often used in MDP software, and their correctness is not guaranteed. In this context, *certificate checkers* can be used to detect bugs in unverified components—such as LP solvers—which we demonstrate with a formally verified certificate checker for linear programming solutions. Finally, we tackled *rounding errors* in floating-point arithmetic by employing directed rounding.

We also demonstrated that implementations based on exact arithmetic can achieve practical performance, particularly in the context of certificate checking. Overall, our work shows that using ITPs for the verification of MDP algorithms is not only feasible but also significantly enhances the reliability of probabilistic model checking and planning software.

Future Work We aim to extend the methodology developed in this thesis to the formalization of more complex probabilistic systems, such as partially observable MDPs, continuous MDPs, and stochastic games. Beyond modeling more expressive systems,

we also plan to study richer classes of properties, e.g. specifications expressed in *linear temporal logic*. Additionally, the technique of directed rounding has potential for broader application, and may be used to devise efficient implementations of further numeric algorithms within Isabelle/HOL.

Another important direction of future work is the elimination of *preprocessing errors* from MDP software. This involves the formal specification of modeling languages for MDPs, as well as the formalization of the transformations required to convert models into representations suitable for efficient computations. Along similar lines, the development of efficient, formally verified parsers for MDP models would be a valuable addition to our pipeline.

Finally, we outline a roadmap to facilitate the practical adoption of our verified algorithms in the probabilistic model checking and planning communities. To gain broader acceptance for verified software, future efforts should prioritize efficient implementations that can serve as drop-in replacements for the equivalent unverified code, or develop certifying algorithms that require minimal changes to existing software. In particular, we aim to support the adoption of certificate-based validation in PMC and planning competitions, requiring unverified software to produce machine-checkable proofs of their results.

6 Publications

This chapter provides summaries of the publications on which this dissertation is based. For each publication, my own contributions are explicitly highlighted. The original versions of the publications are included in the appendix.

Core Publication 1 (Factor of core: 1, sole first author)
Maximilian Schäffeler and Mohammad Abdulaziz. "Formally Verified Solution Methods for Markov Decision Processes." In: *AAAI Conference on Artificial Intelligence*. Vol. 37. 12. AAAI Press, 2023, pp. 15073–15081. DOI: 10.1609/aaai.v37i12.26759

Core Publication 2 (Factor of core: 1, sole first author)
Maximilian Schäffeler and Mohammad Abdulaziz. "Formally Verified Approximate Policy Iteration." In: *AAAI Conference on Artificial Intelligence*. Vol. 39. AAAI Press, 2025. To appear.

Core Publication 3 (Factor of core: 0.5, two first authors)
Bram Kohlen, Maximilian Schäffeler, Mohammad Abdulaziz, Arnd Hartmanns, and Peter Lammich. "A Formally Verified IEEE 754 Floating-Point Implementation of Interval Iteration for MDPs." In: *Computer Aided Verification*. Lecture Notes in Computer Science. Springer, 2025. To appear.

Core Publication 4 (Factor of core: 0.17, six first authors)
Krishnendu Chatterjee, Tim Quatmann, Maximilian Schäffeler, Maximilian Weininger, Tobias Winkler, and Daniel Zilken. "Fixed Point Certificates for Reachability and Expected Rewards in MDPs." In: *Tools and Algorithms for the Construction and Analysis of Systems*. Lecture Notes in Computer Science. Springer, 2025. To appear.

6.1 Core Publication 1

Formally Verified Solution Methods for Markov Decision Processes

Maximilian Schäffeler, Mohammad Abdulaziz

Summary We formally verify executable algorithms for solving Markov decision processes (MDPs) using the interactive theorem prover Isabelle/HOL. As a first step, we develop a formal library of MDPs with rewards, building on existing formalizations of probability theory in Isabelle/HOL and the author's master's thesis. Our analysis focuses on the *expected total reward* criterion for both finite- and infinite-horizon problems.

The core of our formal analysis is the *Bellman optimality operator*: We formally prove that its unique fixed point equals the optimal value of the MDP. Based on these results, we verify several standard algorithms for solving MDPs, including *backward induction*, *value iteration*, *policy iteration*, and the variants *Gauss-Seidel value iteration* and *modified policy iteration*. Our formal convergence proof for the Gauss-Seidel variant fills a gap left in the textbook proof.

We extract executable code from our formalization for all algorithms. To maintain soundness guarantees, all implementations operate over exact arithmetic. We experimentally evaluate the scalability of the verified solvers on standard benchmarks problems. These experiments demonstrate that the verified implementations scale to MDPs with hundreds of thousands of states. Among the verified algorithms, standard value iteration proves to be the most efficient. Hence, we propose an architecture to combine performance with formal guarantees: we pair the verified implementation of value iteration with unverified floating-point planning software. The unverified implementation computes a preliminary solution, which is subsequently refined and validated by the verified solver.

Author Contributions M.S. formalized the algorithms in Isabelle/HOL, performed the formal proofs, generated verified code, and drafted the majority of the paper. M.A. conceived the idea of formally verifying MDP algorithms, performed the benchmarks, and contributed to writing the manuscript. M.S. is the sole first author of this publication.

Maximilian Schäffeler and Mohammad Abdulaziz. "Formally Verified Solution Methods for Markov Decision Processes." In: *AAAI Conference on Artificial Intelligence*. Vol. 37. 12. AAAI Press, 2023, pp. 15073–15081. DOI: 10.1609/aaai.v37i12.26759

6.2 Core Publication 2

Formally Verified Approximate Policy Iteration

Maximilian Schäffeler, Mohammad Abdulaziz

Summary We present a methodology based on interactive theorem proving that facilitates the development of verified implementations of algorithms for solving factored *Markov decision processes* (MDPs). In this work, we focus on the *scalability* of MDP planning algorithms. As a case study, we formally verify an algorithm for *approximate policy iteration* (API) in the proof assistant Isabelle/HOL.

Factored MDPs provide a compact representation of MDPs through structured representations of the state space, as well as the transition and reward functions. As a first step, we formalize a model of factored MDPs in Isabelle/HOL. In this model, transition and reward functions are built from simple components, each of which only observes a subset of the state variables. We also connect this formalization of factored MDPs to our formalization of explicit MDPs.

Based on this foundation, we implement the API algorithm for factored MDPs. API is an iterative method derived from policy iteration that incrementally improves a candidate policy. Each iteration consists of three phases: approximate policy evaluation via linear programming, policy improvement, and Bellman error computation. We formally derive an *a posteriori bound* on the suboptimality of the resulting policy.

We refine the verified algorithm into an executable implementation. As part of this process, we develop verified software for certifying linear programming solutions. We evaluate our implementation on benchmark problems, demonstrating that verified API scales to systems with up to 10^{12} states. The remaining primary scalability bottleneck of the verified implementation lies in the use of the *exact linear programming solvers*.

Author Contributions M.S. formalized the algorithms in Isabelle/HOL, performed the formal proofs, generated verified code, performed the benchmarks, and drafted the majority of the paper. M.A. conceived the idea of formally verifying approximate policy iteration, and contributed to the manuscript. M.S. is the sole first author of this publication.

Maximilian Schäffeler and Mohammad Abdulaziz. "Formally Verified Approximate Policy Iteration." In: *AAAI Conference on Artificial Intelligence*. Vol. 39. AAAI Press, 2025. To appear.

6.3 Core Publication 3

A Formally Verified IEEE 754 Floating-Point Implementation of Interval Iteration for MDPs

Bram Kohlen, Maximilian Schäffeler, Mohammad Abdulaziz, Arnd Hartmanns, Peter Lammich

Summary We present an efficiently executable, formally verified implementation of *interval iteration* for Markov decision processes (MDPs). Our correctness proofs span the entire development pipeline, from high-level abstract semantics of MDPs down to a low-level implementation in LLVM, based on floating-point arithmetic. Using the Isabelle/HOL proof assistant, we verify the convergence of our formal definition of interval iteration, along with the preprocessing transformations required to ensure the algorithm's convergence. Our formal definitions and proofs of correctness are conceptually simpler, yet more precise than those found in existing literature.

We employ *stepwise refinement* to derive an efficient LLVM-based implementation of interval iteration. To that end, we extend the *Isabelle Refinement Framework* with the ability to reason about floating-point arithmetic and directed rounding modes. The extension enables reasoning about a subset of floating-point operations as sound upper and lower bounds of their exact real-valued counterparts. Our approach requires no manual reasoning about floating-point rounding errors. Finally, we experimentally evaluate our implementation on standard benchmarks. The results show that verified interval iteration is *competitive with state-of-the-art tools for MDPs*, while additionally providing formal guarantees on the correctness of the results.

Author Contributions B.K. initiated the collaboration with M.S., both authors drafted the majority of the manuscript together. M.S. formalized the interval iteration algorithm and its preprocessing routines in Isabelle/HOL, and performed the formal correctness proofs. B.K. extended the Isabelle Refinement Framework with support for floating-point arithmetic and refined the abstract algorithm to LLVM. B.K. also evaluated the algorithm on benchmark problems. M.A. and A.H. contributed to the writing, with a focus on the introduction and related work. P.L. embedded the library for floating-point arithmetic with the Isabelle Refinement Framework and contributed to refining the manuscript.

Bram Kohlen et al. "A Formally Verified IEEE 754 Floating-Point Implementation of Interval Iteration for MDPs." In: *Computer Aided Verification*. Lecture Notes in Computer Science. Springer, 2025. To appear.

6.4 Core Publication 4

Fixed Point Certificates for Reachability and Expected Rewards in MDPs

Krishnendu Chatterjee, Tim Quatmann, Maximilian Schäffeler, Maximilian Weininger, Tobias Winkler, Daniel Zilken

Summary The possibility of errors in human-engineered formal verification software, such as model checkers, poses a serious challenge to the reliability and trustworthiness of these tools. In the past, both the algorithms used in probabilistic model checking and their implementations have been sources of soundness issues. An established mitigation strategy to this problem are *certificates*—lightweight, easy-to-check proofs of verification results.

In this paper, we introduce novel certificates for probabilistic model checking of Markov decision processes (MDPs). We focus on *quantitative reachability*, *qualitative reachability* and *expected reward* properties. Our approach is conceptually simple and relies almost exclusively on elementary fixed point theory. The certificates apply to arbitrary finite MDPs and can be efficiently generated with minimal overhead using standard algorithms.

We formally verify the soundness of our certificates using the Isabelle/HOL proof assistant and derive a formally verified certificate checker. Furthermore, we augment existing algorithms in the probabilistic model checker Storm with the ability to produce these certificates. We demonstrate the practicality of our approach by conducting the first formal certification of reference results from the *Quantitative Verification Benchmark Set*. Our approach is a contribution towards integrating certified verification results into safety-critical systems and probabilistic model checking competitions.

Author Contributions M.W. and T.W. initiated the project, drafted the majority of the manuscript and developed major parts of the theory together with D.Z.. T.Q. focused on the implementation in Storm, the certificate generation, and the experimental evaluation. T.Q. also introduced the complementary distance operator. K.C. provided advice on extending the theory to more general settings. M.S. formalized the theory of fixed point certificates in Isabelle/HOL, refining several proofs in the process and wrote the section on the formalization. M.S. derived the verified executable certificate checker from the formalization. The authors are listed in alphabetical order.

Krishnendu Chatterjee et al. "Fixed Point Certificates for Reachability and Expected Rewards in MDPs." In: *Tools and Algorithms for the Construction and Analysis of Systems*. Lecture Notes in Computer Science. Springer, 2025. To appear.

Bibliography

[1] Naeem Abbasi. "Formal Reliability Analysis Using Higher-order Logic Theorem Proving." PhD thesis. Concordia University, 2012. 161 pp.

[2] Rosa Abbasi, Jonas Schiffl, Eva Darulova, Mattias Ulbrich, and Wolfgang Ahrendt. "Deductive Verification of Floating-Point Java Programs in KeY." In: *Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2021, pp. 242–261. DOI: 10.1007/978-3-030-72013-1_13.

[3] Mohammad Abdulaziz, Charles Gretton, and Michael Norrish. "A Verified Compositional Algorithm for AI Planning." In: *Interactive Theorem Proving*. Vol. 141. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019, 4:1–4:19. DOI: 10.4230/LIPIcs.ITP.2019.4.

[4] Mohammad Abdulaziz and Lukas Koller. "Formal Semantics and Formally Verified Validation for Temporal Planning." In: *AAAI Conference on Artificial Intelligence*. Vol. 36. 2022, pp. 9635–9643. DOI: 10.1609/aaai.v36i9.21197.

[5] Mohammad Abdulaziz and Friedrich Kurz. "Formally Verified SAT-Based AI Planning." In: *AAAI Conference on Artificial Intelligence*. Vol. 37. 2023, pp. 14665–14673. DOI: 10.1609/aaai.v37i12.26714.

[6] Mohammad Abdulaziz and Peter Lammich. "A Formally Verified Validator for Classical Planning Problems and Solutions." In: *International Conference on Tools with Artificial Intelligence*. IEEE, 2018, pp. 474–479. DOI: 10.1109/ICTAI.2018.00079.

[7] Reynald Affeldt and Cyril Cohen. "Measure Construction by Extension in Dependent Type Theory with Application to Integration." In: *Journal of Automated Reasoning* 67.3 (2023), pp. 1–27. DOI: 10.1007/s10817-023-09671-5.

[8] Reynald Affeldt, Manabu Hagiwara, and Jonas Sénizergues. "Formalization of Shannon's Theorems." In: *Journal of Automated Reasoning* 53.1 (2014), pp. 63–103. DOI: 10.1007/S10817-013-9298-1.

[9] Eyad Alkassar, Sascha Böhme, Kurt Mehlhorn, and Christine Rizkallah. "Verification of Certifying Computations." In: *Computer Aided Verification*. Springer, 2011, pp. 67–82. DOI: 10.1007/978-3-642-22110-1_7.

[10] David L. Applegate, William Cook, Sanjeeb Dash, and Daniel G. Espinoza. "Exact Solutions to Linear Programming Problems." In: *Operations Research Letters* 35.6 (2007), pp. 693–699. DOI: 10.1016/j.orl.2006.12.010.

[11] Philippe Audebaud and Christine Paulin-Mohring. "Proofs of Randomized Algorithms in Coq." In: *Science of Computer Programming* 74.8 (2009), pp. 568–589. DOI: 10.1016/j.scico.2007.09.002.

[12] Jeremy Avigad, Johannes Hölzl, and Luke Serafin. "A Formally Verified Proof of the Central Limit Theorem." In: *Journal of Automated Reasoning* 59.4 (2017), pp. 389–423. DOI: 10.1007/S10817-017-9404-X.

[13] Alexander Bagnall and Gordon Stewart. "Certifying the True Error: Machine Learning in Coq with Verified Generalization Guarantees." In: *AAAI Conference on Artificial Intelligence*. Vol. 33. 2019, pp. 2662–2669. DOI: 10.1609/aaai.v33i01.33012662.

[14] Christel Baier, Luca de Alfaro, Vojtech Forejt, and Marta Kwiatkowska. "Model Checking Probabilistic Systems." In: *Handbook of Model Checking*. Springer, 2018, pp. 963–999.

[15] Christel Baier, Calvin Chau, and Sascha Klüppelholz. "Certificates and Witnesses for Multi-objective Queries in Markov Decision Processes." In: *Quantitative Evaluation of Systems and Formal Modeling and Analysis of Timed Systems*. Springer, 2024, pp. 1–18. DOI: 10.1007/978-3-031-68416-6_1.

[16] Christel Baier, Boudewijn R. Haverkort, Holger Hermanns, and Joost-Pieter Katoen. "Performance Evaluation and Model Checking Join Forces." In: *Communications of the ACM* 53.9 (2010), pp. 76–85. DOI: 10.1145/1810891.1810912.

[17] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 2008.

[18] Christel Baier, Joachim Klein, Linda Leuschner, David Parker, and Sascha Wunderlich. "Ensuring the Reliability of Your Model Checker: Interval Iteration for Markov Decision Processes." In: *Computer Aided Verification*. Springer, 2017, pp. 160–180. DOI: 10.1007/978-3-319-63387-9_8.

[19] Tomas Balyo, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda, eds. *Proceedings of SAT Competition 2023: Solver, Benchmark and Proof Checker Descriptions*. Department of Computer Science Series of Publications B. University of Helsinki, 2023.

[20] Grzegorz Bancerek, Czesław Byliński, Adam Grabowski, Artur Korniłowicz, Roman Matuszewski, Adam Naumowicz, Karol Pak, and Josef Urban. "Mizar: State-of-the-art and Beyond." In: *Intelligent Computer Mathematics*. Springer, 2015, pp. 261–279. DOI: 10.1007/978-3-319-20615-8_17.

[21] Kshitij Bansal, Sarah M. Loos, Markus N. Rabe, Christian Szegedy, and Stewart Wilcox. "HOList: An Environment for Machine Learning of Higher Order Logic Theorem Proving." In: *International Conference on Machine Learning*. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 454–463.

[22] Geoff Barrett. "Formal Methods Applied to a Floating-Point Number System." In: *IEEE Transactions on Software Engineering* 15.5 (1989), pp. 611–621. DOI: 10.1109/32.24710.

[23] Gilles Barthe, François Dupressoir, Benjamin Grégoire, César Kunz, Benedikt Schmidt, and Pierre-Yves Strub. "EasyCrypt: A Tutorial." In: *Foundations of Security Analysis and Design*. Springer, 2014, pp. 146–166. DOI: 10.1007/978-3-319-10082-1_6.

[24] Gilles Barthe, Benjamin Grégoire, Sylvain Heraud, and Santiago Zanella Béguelin. "Computer-Aided Security Proofs for the Working Cryptographer." In: *Advances in Cryptology*. Annual Cryptology Conference. Springer, 2011, pp. 71–90. DOI: 10.1007/978-3-642-22792-9_5.

[25] David A. Basin, Andreas Lochbihler, and S. Reza Sefidgar. "CryptHOL: Game-Based Proofs in Higher-Order Logic." In: *Journal of Cryptology* 33.2 (2020), pp. 494–566. DOI: 10.1007/S00145-019-09341-Z.

[26] Heiko Becker, Nikita Zyuzin, Raphaël Monat, Eva Darulova, Magnus O. Myreen, and Anthony C. J. Fox. "A Verified Certificate Checker for Finite-Precision Error Bounds in Coq and HOL4." In: *Formal Methods in Computer Aided Design*. IEEE, 2018, pp. 1–10.

[27] Alexander Bentkamp, Jasmin Christian Blanchette, and Dietrich Klakow. "A Formal Proof of the Expressiveness of Deep Learning." In: *Journal of Automated Reasoning* 63.2 (2019), pp. 347–368. DOI: 10.1007/s10817-018-9481-5.

[28] Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development*. Red. by Wilfried Brauer, Grzegorz Rozenberg, and Arto Salomaa. Texts in Theoretical Computer Science. Springer, 2004. DOI: 10.1007/978-3-662-07964-5.

[29] Dirk Beyer. "Competition on Software Verification and Witness Validation: SV-COMP 2023." In: *Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2023, pp. 495–522. DOI: 10.1007/978-3-031-30820-8_29.

[30] Józef Białas. "The σ-Additive Measure Theory." In: *Formalized Mathematics* 2.2 (1991), pp. 263–270.

[31] Jan Olaf Blech and Benjamin Grégoire. "Certifying Compilers Using Higher-Order Theorem Provers as Certificate Checkers." In: *Formal Methods in System Design* 38.1 (2011), pp. 33–61. DOI: 10.1007/s10703-010-0108-7.

[32] Sylvie Boldo, Jacques-Henri Jourdan, Xavier Leroy, and Guillaume Melquiond. "Verified Compilation of Floating-Point Computations." In: *Journal of Automated Reasoning* 54.2 (2015), pp. 135–163. DOI: 10.1007/s10817-014-9317-x.

[33] Sylvie Boldo and Guillaume Melquiond. "Flocq: A Unified Library for Proving Floating-Point Algorithms in Coq." In: *IEEE Symposium on Computer Arithmetic*. 2011, pp. 243–252. DOI: 10.1109/ARITH.2011.40.

[34] Sylvie Boldo and Cesar Munoz. *A High-Level Formalization of Floating-Point Number in PVS*. 2006. URL: https://shemesh.larc.nasa.gov/fm/papers/float.pdf (visited on 2025-04-09).

[35] Suresh Bolusani, Mathieu Besançon, Ksenia Bestuzheva, Antonia Chmiela, João Dionísio, Tim Donkiewicz, Jasper van Doornmalen, Leon Eifler, Mohammed Ghannam, Ambros Gleixner, Christoph Graczyk, Katrin Halbig, Ivo Hedtke, Alexander Hoen, Christopher Hojny, Rolf van der Hulst, Dominik Kamp, Thorsten Koch, Kevin Kofler, Jurgen Lentz, Julian Manns, Gioni Mexi, Erik Mühmer, Marc E. Pfetsch, Franziska Schlösser, Felipe Serrano, Yuji Shinano, Mark Turner, Stefan Vigerske, Dieter Weninger, and Lixing Xu. *The SCIP Optimization Suite 9.0*. Technical Report. Optimization Online, 2024. URL: https://optimization-online.org/2024/02/the-scip-optimization-suite-9-0/ (visited on 2025-04-09).

[36] Carlos E. Budde, Christian Dehnert, Ernst Moritz Hahn, Arnd Hartmanns, Sebastian Junges, and Andrea Turrini. "JANI: Quantitative Model and Tool Interaction." In: *Tools and Algorithms for the Construction and Analysis of Systems*. Vol. 10206. Lecture Notes in Computer Science. 2017, pp. 151–168. DOI: 10.1007/978-3-662-54580-5_9.

[37] Orieta Celiku and Annabelle McIver. "Cost-Based Analysis of Probabilistic Programs Mechanised in HOL." In: *Nordic Journal of Computing* 11.2 (2004), pp. 102–128.

[38] Krishnendu Chatterjee, Tim Quatmann, Maximilian Schäffeler, Maximilian Weininger, Tobias Winkler, and Daniel Zilken. "Fixed Point Certificates for Reachability and Expected Rewards in MDPs." In: *Tools and Algorithms for the Construction and Analysis of Systems*. Lecture Notes in Computer Science. Springer, 2025. To appear.

[39] Mark Chevallier and Jacques D. Fleuriot. *Formalising the Foundations of Discrete Reinforcement Learning in Isabelle/HOL*. 2021. DOI: 10.48550/arXiv.2112.05996. arXiv: 2112.05996. Pre-published.

[40] Aaron Richard Coble. "Anonymity, Information, and Machine-Assisted Proof." PhD thesis. University of Cambridge, UK, 2010.

[41] David A. Cock. "Verifying Probabilistic Correctness in Isabelle with pGCL." In: *Systems Software Verification*. Vol. 102. EPTCS. 2012, pp. 167–178. DOI: 10.4204/EPTCS.102.15.

[42] The mathlib Community. "The Lean Mathematical Library." In: *Certified Programs and Proofs*. Association for Computing Machinery, 2020, pp. 367–381. DOI: 10.1145/3372885.3373824.

[43] Patrick Cousot, Radhia Cousot, Jerôme Feret, Laurent Mauborgne, Antoine Miné, David Monniaux, and Xavier Rival. "The ASTREÉ Analyzer." In: *Programming Languages and Systems*. European Symposium on Programming. Springer, 2005, pp. 21–30. DOI: 10.1007/978-3-540-31987-0_3.

[44] Pedro R. D'Argenio, Bertrand Jeannet, Henrik E. Jensen, and Kim G. Larsen. "Reachability Analysis of Probabilistic Systems by Successive Refinements." In: *Process Algebra and Probabilistic Methods. Performance Modelling and Verification*. Springer, 2001, pp. 39–56. DOI: 10.1007/3-540-44804-7_3.

[45] Matthew L. Daggitt, Wen Kokke, Robert Atkey, Luca Arnaboldi, and Ekaterina Komendantskya. *Vehicle: Interfacing Neural Network Verifiers with Interactive Theorem Provers*. 2022. DOI: 10.48550/arXiv.2202.05207. arXiv: 2202.05207 [cs]. Pre-published.

[46] Marc Daumas and David R. Lester. "Stochastic Formal Methods: An Application to Accuracy of Numeric Software." In: *Hawaii International International Conference on Systems Science*. IEEE, 2007, p. 262. DOI: 10.1109/HICSS.2007.499.

[47] Marc Daumas, David R. Lester, Érik Martin-Dorel, and Annick Truffert. "Improved Bound for Stochastic Formal Correctness of Numerical Algorithms." In: *Innovations in Systems and Software Engineering* 6.3 (2010), pp. 173–179. DOI: 10.1007/S11334-010-0128-X.

[48] Florent de Dinechin, Christoph Lauter, and Guillaume Melquiond. "Certifying the Floating-Point Implementation of an Elementary Function Using Gappa." In: *IEEE Transactions on Computers* 60.2 (2011), pp. 242–253. DOI: 10.1109/TC.2010.128.

[49] Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. "The Lean Theorem Prover (System Description)." In: *Automated Deduction*. Springer, 2015, pp. 378–388. DOI: 10.1007/978-3-319-21401-6_26.

[50] Agnes Doll. "Kolmogorov's Zero-One Law." In: *Formalized Mathematics* 17.2 (2009), pp. 73–77. DOI: 10.2478/v10037-009-0008-8.

[51] Manuel Eberl, Max W. Haslbeck, and Tobias Nipkow. "Verified Analysis of Random Binary Tree Structures." In: *Journal of Automated Reasoning* 64.5 (5 2020), pp. 879–910. DOI: 10.1007/s10817-020-09545-0.

[52] Manuel Eberl, Johannes Hölzl, and Tobias Nipkow. "A Verified Compiler for Probability Density Functions." In: *Programming Languages and Systems*. Springer, 2015, pp. 80–104. DOI: 10.1007/978-3-662-46669-8_4.

[53] Mnacho Echenim, Hervé Guiol, and Nicolas Peltier. "Formalizing the Cox–Ross–Rubinstein Pricing of European Derivatives in Isabelle/HOL." In: *Journal of Automated Reasoning* 64.4 (2020), pp. 737–765. DOI: 10.1007/s10817-019-09528-w.

[54] Mnacho Echenim and Nicolas Peltier. "The Binomial Pricing Model in Finance: A Formalization in Isabelle." In: *Automated Deduction*. Springer, 2017, pp. 546–562. DOI: 10.1007/978-3-319-63046-5_33.

[55] Burak Ekici, Alain Mebsout, Cesare Tinelli, Chantal Keller, Guy Katz, Andrew Reynolds, and Clark Barrett. "SMTCoq: A Plug-in for Integrating SMT Solvers into Coq." In: *Computer Aided Verification*. 2017. DOI: 10.1007/978-3-319-63390-9_7.

[56] Noboru Endou, Keiko Narita, and Yasunari Shidama. "The Lebesgue Monotone Convergence Theorem." In: *Formalized Mathematics* 16.2 (2008), pp. 167–175. DOI: 10.2478/v10037-008-0023-1.

[57] Salomé Eriksson, Gabriele Röger, and Malte Helmert. "A Proof System for Unsolvable Planning Tasks." In: *International Conference on Automated Planning and Scheduling* 28 (2018), pp. 65–73. DOI: 10.1609/icaps.v28i1.13899.

[58] Salomé Eriksson, Gabriele Röger, and Malte Helmert. "Unsolvability Certificates for Classical Planning." In: *International Conference on Automated Planning and Scheduling*. Vol. 27. 2017, pp. 88–97. DOI: 10.1609/icaps.v27i1.13818.

[59] Javier Esparza, Peter Lammich, René Neumann, Tobias Nipkow, Alexander Schimpf, and Jan-Georg Smaus. "A Fully Verified Executable LTL Model Checker." In: *Computer Aided Verification*. Springer, 2013, pp. 463–478. DOI: 10.1007/978-3-642-39799-8_31.

[60] Florian Funke, Simon Jantsch, and Christel Baier. "Farkas Certificates and Minimal Witnesses for Probabilistic Reachability Constraints." In: *Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2020, pp. 324–345. DOI: 10.1007/978-3-030-45190-5_18.

[61] Google, AlphaProof and AlphaGeometry teams. *AI Achieves Silver-Medal Standard Solving International Mathematical Olympiad Problems*. 2024. URL: https://deepmind.google/discover/blog/ai-solves-imo-problems-at-silver-medal-level/ (visited on 2025-04-09).

[62] Eric Goubault and Sylvie Putot. "Static Analysis of Finite Precision Computations." In: *Verification, Model Checking, and Abstract Interpretation*. Springer, 2011, pp. 232–247. DOI: 10.1007/978-3-642-18275-4_17.

[63] C. Guestrin, D. Koller, R. Parr, and S. Venkataraman. "Efficient Solution Algorithms for Factored MDPs." In: *Journal of Artificial Intelligence Research* 19 (2003), pp. 399–468. DOI: 10.1613/jair.1000.

[64] Serge Haddad and Benjamin Monmege. "Interval Iteration Algorithm for MDPs and IMDPs." In: *Theoretical Computer Science*. Reachability Problems 2014: Special Issue 735 (2018), pp. 111–131. DOI: 10.1016/j.tcs.2016.12.003.

[65] Florian Haftmann and Tobias Nipkow. "Code Generation via Higher-Order Rewrite Systems." In: *Functional and Logic Programming*. Springer, 2010, pp. 103–117. DOI: 10.1007/978-3-642-12251-4_9.

[66] Tingting Han and Joost-Pieter Katoen. "Counterexamples in Probabilistic Model Checking." In: *Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2007, pp. 72–86. DOI: 10.1007/978-3-540-71209-1_8.

[67] J. Harrison. "Formal Verification at Intel." In: *Logic in Computer Science*. 2003, pp. 45–54. DOI: 10.1109/LICS.2003.1210044.

[68] John Harrison. "A Machine-Checked Theory of Floating Point Arithmetic." In: *Theorem Proving in Higher Order Logics*. Springer, 1999, pp. 113–130. DOI: 10.1007/3-540-48256-3_9.

[69] John Harrison. "Floating-Point Verification Using Theorem Proving." In: *Formal Methods for Hardware Verification*. International School on Formal Methods for the Design of Computer, Communication and Software Systems. Springer, 2006, pp. 211–242. DOI: 10.1007/11757283_8.

[70] Arnd Hartmanns. "Correct Probabilistic Model Checking with Floating-Point Arithmetic." In: *Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2022, pp. 41–59. DOI: 10.1007/978-3-030-99527-0_3.

[71] Arnd Hartmanns and Holger Hermanns. "The Modest Toolset: An Integrated Environment for Quantitative Modelling and Verification." In: *Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2014, pp. 593–598. DOI: 10.1007/978-3-642-54862-8_51.

[72] Arnd Hartmanns, Sebastian Junges, Tim Quatmann, and Maximilian Weininger. "A Practitioner's Guide to MDP Model Checking Algorithms." In: *Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2023, pp. 469–488. DOI: 10.1007/978-3-031-30823-9_24.

[73] Arnd Hartmanns and Benjamin Lucien Kaminski. "Optimistic Value Iteration." In: *Computer Aided Verification*. Springer, 2020, pp. 488–511. DOI: 10.1007/978-3-030-53291-8_26.

[74] Arnd Hartmanns, Michaela Klauck, David Parker, Tim Quatmann, and Enno Ruijters. "The Quantitative Verification Benchmark Set." In: *Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2019, pp. 344–350. DOI: 10.1007/978-3-030-17462-0_20.

[75] Arnd Hartmanns, Bram Kohlen, and Peter Lammich. "Efficient Formally Verified Maximal End Component Decomposition for MDPs." In: *Formal Methods*. Ed. by André Platzer, Kristin Yvonne Rozier, Matteo Pradella, and Matteo Rossi. Vol. 14933. Lecture Notes in Computer Science. Springer, 2024, pp. 206–225. DOI: 10.1007/978-3-031-71162-6_11.

[76] Arnd Hartmanns, Bram Kohlen, and Peter Lammich. "Fast Verified SCCs for Probabilistic Model Checking." In: *Automated Technology for Verification and Analysis*. Ed. by Étienne André and Jun Sun. Vol. 14215. Lecture Notes in Computer Science. Springer, 2023, pp. 181–202. DOI: 10.1007/978-3-031-45329-8_9.

[77] Osman Hasan. "Formal Probabilistic Analysis Using Theorem Proving." PhD thesis. Concordia University, 2008.

[78] Nicholas A. J. Hastings. "The Repair Limit Replacement Method." In: *Journal of the Operational Research Society* 20.3 (1969), pp. 337–349. DOI: 10.1057/jors.1969.78.

[79] Christian Hensel, Sebastian Junges, Joost-Pieter Katoen, Tim Quatmann, and Matthias Volk. "The Probabilistic Model Checker Storm." In: *International Journal on Software Tools for Technology Transfer* 24.4 (2022), pp. 589–610. DOI: 10.1007/s10009-021-00633-z.

[80] Alasdair Hill, Ekaterina Komendantskaya, and Ronald P. A. Petrick. "Proof-Carrying Plans: A Resource Logic for AI Planning." In: *Proceedings of the 22nd International Symposium on Principles and Practice of Declarative Programming*. PPDP '20. Association for Computing Machinery, 2020-09-21, pp. 1–13. DOI: 10.1145/3414080.3414094.

[81] Johannes Hölzl. "Construction and Stochastic Applications of Measure Spaces in Higher-Order Logic." Technische Universität München, 2013.

[82] Johannes Hölzl. "Formalising Semantics for Expected Running Time of Probabilistic Programs." In: *Interactive Theorem Proving*. Springer, 2016, pp. 475–482. DOI: 10.1007/978-3-319-43144-4_30.

[83] Johannes Hölzl. "Markov Chains and Markov Decision Processes in Isabelle/HOL." In: *Journal of Automated Reasoning* 59.3 (2017), pp. 345–387. DOI: 10.1007/s10817-016-9401-5.

[84] Johannes Hölzl and Armin Heller. "Three Chapters of Measure Theory in Isabelle/HOL." In: *Interactive Theorem Proving*. Springer, 2011, pp. 135–151. DOI: 10.1007/978-3-642-22863-6_12.

[85] Johannes Hölzl, Andreas Lochbihler, and Dmitriy Traytel. "A Formalized Hierarchy of Probabilistic System Types." In: *Interactive Theorem Proving*. Springer, 2015, pp. 203–220. DOI: 10.1007/978-3-319-22102-1_13.

[86] Johannes Hölzl and Tobias Nipkow. "Interactive Verification of Markov Chains: Two Distributed Protocol Case Studies." In: *Quantities in Formal Methods*. Vol. 103. EPTCS. 2012, pp. 17–31. DOI: 10.4204/EPTCS.103.2.

[87] Johannes Hölzl and Tobias Nipkow. "Verifying pCTL Model Checking." In: *Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2012, pp. 347–361. DOI: 10.1007/978-3-642-28756-5_24.

[88] Daniel Huang, Prafulla Dhariwal, Dawn Song, and Ilya Sutskever. *GamePad: A Learning Environment for Theorem Proving*. 2018. DOI: 10.48550/arXiv.1806.00608. arXiv: 1806.00608 [cs]. Pre-published.

[89] Joe Hurd. *Formal Verification of Probabilistic Algorithms*. UCAM-CL-TR-566. University of Cambridge, Computer Laboratory, 2003.

[90] Joe Hurd, Annabelle McIver, and Carroll Morgan. "Probabilistic Guarded Commands Mechanized in *HOL*." In: *Theoretical Computer Science*. Quantitative Aspects of Programming Languages 346.1 (2005), pp. 96–112. DOI: 10.1016/j.tcs.2005.08.005.

[91] Peter Jaeger. "Borel-Cantelli Lemma." In: *Formalized Mathematics* 19.4 (2012), pp. 227–232. DOI: 10.2478/v10037-011-0031-4.

[92] Peter Jaeger. "Modelling Real World Using Stochastic Processes and Filtration." In: *Formalized Mathematics* 24.1 (2016), pp. 1–16. DOI: 10.1515/forma-2016-0001.

[93] Jan Jakubuv and Josef Urban. "Hammering Mizar by Learning Clause Guidance (Short Paper)." In: *Interactive Theorem Proving*. Vol. 141. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 34:1–34:8.

[94] Simon Jantsch. "Certificates and Witnesses for Probabilistic Model Checking." 2022.

[95] Simon Jantsch, Hans Harder, Florian Funke, and Christel Baier. "Switss: Computing Small Witnessing Subsystems." In: *Formal Methods in Computer Aided Design*. IEEE, 2020, pp. 236–244. DOI: 10.34727/2020/ISBN.978-3-85448-042-6_31.

[96] Cezary Kaliszyk, François Chollet, and Christian Szegedy. "HolStep: A Machine Learning Dataset for Higher-order Logic Theorem Proving." In: *International Conference on Learning Representations*. 2017.

[97] Cezary Kaliszyk, Josef Urban, Henryk Michalewski, and Miroslav Olsák. "Reinforcement Learning of Theorem Proving." In: *Advances in Neural Information Processing Systems*. 2018, pp. 8836–8847.

[98] Mojgan Kamali and Joost-Pieter Katoen. "Probabilistic Model Checking of AODV." In: *Quantitative Evaluation of Systems*. Springer, 2020, pp. 54–73. DOI: 10.1007/978-3-030-59854-9_6.

[99] Florian Kammüller, Markus Wenzel, and Lawrence C. Paulson. "Locales A Sectioning Concept for Isabelle." In: *Theorem Proving in Higher Order Logics*. Springer, 1999, pp. 149–165. DOI: 10.1007/3-540-48256-3_11.

[100] Ata Keskin. "Martingales." In: *Archive of Formal Proofs* (2023). URL: https://www.isa-afp.org/entries/Martingales.html.

[101] Florent Kirchner, Nikolai Kosmatov, Virgile Prevosto, Julien Signoles, and Boris Yakobowski. "Frama-C: A Software Analysis Perspective." In: *Formal Aspects of Computing* 27.3 (2015), pp. 573–609. DOI: 10.1007/s00165-014-0326-7.

[102] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. "seL4: Formal Verification of an OS Kernel." In: *Symposium on Operating Systems Principles*. Association for Computing Machinery, 2009, pp. 207–220. DOI: 10.1145/1629575.1629596.

[103] Bram Kohlen, Maximilian Schäffeler, Mohammad Abdulaziz, Arnd Hartmanns, and Peter Lammich. "A Formally Verified IEEE 754 Floating-Point Implementation of Interval Iteration for MDPs." In: *Computer Aided Verification*. Lecture Notes in Computer Science. Springer, 2025. To appear.

[104] Nikolai Kosmatov, Virgile Prevosto, and Julien Signoles, eds. *Guide to Software Verification with Frama-C: Core Components, Usages, and Applications*. Computer Science Foundations and Applied Logic. Springer, 2024. DOI: 10.1007/978-3-031-55608-1.

[105] Dieter Kratsch, Ross M. McConnell, Kurt Mehlhorn, and Jeremy P. Spinrad. "Certifying Algorithms for Recognizing Interval Graphs and Permutation Graphs." In: *ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, 2003, pp. 158–167.

[106] Katharina Kreuzer. "Verification of Correctness and Security Properties for CRYSTALS-KYBER." In: *Computer Security Foundations Symposium*. IEEE, 2024, pp. 511–526. DOI: 10.1109/CSF61375.2024.00016.

[107] Orna Kupferman and Moshe Y. Vardi. "From Complementation to Certification." In: *Theoretical Computer Science*. Tools and Algorithms for the Construction and Analysis of Systems 345.1 (2005), pp. 83–100. DOI: 10.1016/j.tcs.2005.07.021.

[108] Marta Kwiatkowska, Gethin Norman, and David Parker. "PRISM 4.0: Verification of Probabilistic Real-Time Systems." In: *Computer Aided Verification*. Lecture Notes in Computer Science. Springer, 2011, pp. 585–591. DOI: 10.1007/978-3-642-22110-1_47.

[109] Peter Lammich. "Fast and Verified UNSAT Certificate Checking." In: *Automated Reasoning*. Springer, 2024, pp. 439–457. DOI: 10.1007/978-3-031-63498-7_26.

[110] Peter Lammich. "Refinement to Imperative HOL." In: *Journal of Automated Reasoning* 62.4 (4 2019), pp. 481–503. DOI: 10.1007/s10817-017-9437-1.

[111] Xavier Leroy. "Formal Verification of a Realistic Compiler." In: *Communications of the ACM* 52.7 (2009), pp. 107–115. DOI: 10.1145/1538788.1538814.

[112] David R Lester. "Topology in PVS: Continuous Mathematics with Applications." In: *Second Workshop on Automated Formal Methods*. Association for Computing Machinery, 2007, pp. 11–20. DOI: 10.1145/1345169.1345171.

[113] Victor Magron, George Constantinides, and Alastair Donaldson. "Certified Roundoff Error Bounds Using Semidefinite Programming." In: *ACM Trans. Math. Softw.* 43.4 (2017), 34:1–34:31. DOI: 10.1145/3015465.

[114] *Mathlib4/Mathlib/Probability at Master · Leanprover-Community/Mathlib4*. URL: https://github.com/leanprover-community/mathlib4/tree/master/Mathlib/Probability (visited on 2025-04-09).

[115] R. M. McConnell, K. Mehlhorn, S. Näher, and P. Schweitzer. "Certifying Algorithms." In: *Computer Science Review* 5.2 (2011), pp. 119–161. DOI: 10.1016/j.cosrev.2010.09.009.

[116] Franz Merkl. "Dynkin's Lemma in Measure Theory." In: *Journal of Formalized Mathematics* 9.3 (2001), pp. 591–595.

[117] Tarek Mhamdi, Osman Hasan, and Sofiène Tahar. "Formalization of Entropy Measures in HOL." In: *Interactive Theorem Proving*. Springer, 2011, pp. 233–248. DOI: 10.1007/978-3-642-22863-6_18.

[118] Tarek Mhamdi, Osman Hasan, and Sofiène Tahar. "On the Formalization of the Lebesgue Integration Theory in HOL." In: *Interactive Theorem Proving*. Springer, 2010, pp. 387–402. DOI: 10.1007/978-3-642-14052-5_27.

[119] Paul S. Miner. *Defining the IEEE-854 Floating-Point Standard in PVS*. NASA Langley Technical Report Server, 1995.

[120] Kedar S. Namjoshi. "Certifying Model Checkers." In: *Computer Aided Verification*. Computer Aided Verification. Springer, 2001, pp. 2–13. DOI: 10.1007/3-540-44585-4_2.

[121] Andrzej Nędzusiak. "Probability." In: *Journal of Formalized Mathematics* 1.4 (1990), pp. 745–749.

[122] Thi Minh Tuyen Nguyen and Claude Marché. "Hardware-Dependent Proofs of Numerical Programs." In: *Certified Programs and Proofs*. Springer, 2011, pp. 314–329. DOI: 10.1007/978-3-642-25379-9_23.

[123] Tobias Nipkow, Manuel Eberl, and Maximilian P. L. Haslbeck. "Verified Textbook Algorithms: A Biased Survey." In: *Automated Technology for Verification and Analysis*. Springer, 2020, pp. 25–53. DOI: 10.1007/978-3-030-59152-6_2.

[124] Tobias Nipkow, Markus Wenzel, and Lawrence C. Paulson, eds. *Isabelle/HOL*. Red. by Gerhard Goos, Juris Hartmanis, and Jan Van Leeuwen. Vol. 2283. Lecture Notes in Computer Science. Springer, 2002. DOI: 10.1007/3-540-45949-9.

[125] Lars Noschinski. "Proof Pearl: A Probabilistic Proof for the Girth-Chromatic Number Theorem." In: *Interactive Theorem Proving*. Vol. 7406. Lecture Notes in Computer Science. Springer, 2012, pp. 393–404. DOI: 10.1007/978-3-642-32347-8_27.

[126] Lars Noschinski, Christine Rizkallah, and Kurt Mehlhorn. "Verification of Certifying Computations through AutoCorres and Simpl." In: *NASA Formal Methods*. Springer, 2014, pp. 46–61. DOI: 10.1007/978-3-319-06200-6_4.

[127] Steven Obua. "Proving Bounds for Real Linear Programs in Isabelle/HOL." In: *Theorem Proving in Higher Order Logics*. Springer, 2005, pp. 227–244. DOI: 10.1007/11541868_15.

[128] Hiroyuki Okazaki. "Probability on Finite and Discrete Set and Uniform Distribution." In: *Formalized Mathematics* 17.2 (2009), pp. 173–178. DOI: 10.2478/v10037-009-0020-z.

[129] Hiroyuki Okazaki and Yasunari Shidama. "Probability Measure on Discrete Spaces and Algebra of Real-Valued Random Variables." In: *Formalized Mathematics* 18.4 (2011), pp. 213–217. DOI: 10.2478/v10037-010-0026-6.

[130] S. Owre, J. M. Rushby, and N. Shankar. "PVS: A Prototype Verification System." In: *Automated Deduction*. Springer, 1992, pp. 748–752. DOI: 10.1007/3-540-55602-8_217.

[131] Lawrence C. Paulson and Jasmin Christian Blanchette. "Three Years of Experience with Sledgehammer, a Practical Link Between Automatic and Interactive Theorem Provers." In: The 8th International Workshop on the Implementation of Logics, pp. 1–11. DOI: 10.29007/36dt.

[132] Doron Peled, Amir Pnueli, and Lenore Zuck. "From Falsification to Verification." In: *Foundations of Software Technology and Theoretical Computer Science*. Springer, 2001, pp. 292–304. DOI: 10.1007/3-540-45294-X_25.

[133] Andrei Popescu, Johannes Hölzl, and Tobias Nipkow. "Formalizing Probabilistic Noninterference." In: *Certified Programs and Proofs*. Springer, 2013, pp. 259–275. DOI: 10.1007/978-3-319-03545-1_17.

[134] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. 1st ed. Wiley Series in Probability and Statistics. Wiley, 1994. DOI: 10.1002/9780470316887.

[135] Muhammad Qasim, Osman Hasan, Maissa Elleuch, and Sofiène Tahar. "Formalization of Normal Random Variables in HOL." In: *Intelligent Computer Mathematics*. Springer, 2016, pp. 44–59. DOI: 10.1007/978-3-319-42547-4_4.

[136] Tim Quatmann and Joost-Pieter Katoen. "Sound Value Iteration." In: *Computer Aided Verification*. Springer, 2018, pp. 643–661. DOI: 10.1007/978-3-319-96145-3_37.

[137] Tahina Ramananandro, Paul Mountcastle, Benoît Meister, and Richard Lethin. "A Unified Coq Framework for Verifying C Programs with Floating-Point Computations." In: *Certified Programs and Proofs*. Association for Computing Machinery, 2016, pp. 15–26. DOI: 10.1145/2854065.2854066.

[138] Stefan Richter. "Formalizing Integration Theory with an Application to Probabilistic Algorithms." In: *Theorem Proving in Higher Order Logics*. Vol. 3223. Lecture Notes in Computer Science. Springer, 2004, pp. 271–286. DOI: 10.1007/978-3-540-30142-4_20.

[139] Riley Roberts, Benjamin Lewis, Arnd Hartmanns, Prabal Basu, Sanghamitra Roy, Koushik Chakraborty, and Zhen Zhang. "Probabilistic Verification for Reliability of a Two-by-Two Network-on-Chip System." In: *Formal Methods for Industrial Critical Systems*. Springer, 2021, pp. 232–248. DOI: 10.1007/978-3-030-85248-1_16.

[140] Enno Ruijters, Dennis Guck, Peter Drolenga, Margot Peters, and Mariëlle Stoelinga. "Maintenance Analysis and Optimization via Statistical Model Checking." In: *Quantitative Evaluation of Systems*. Springer, 2016, pp. 331–347. DOI: 10.1007/978-3-319-43425-4_22.

[141] Enno Ruijters, Dennis Guck, Martijn Van Noort, and Mariëlle Stoelinga. "Reliability-Centered Maintenance of the Electrically Insulated Railway Joint via Fault Tree Analysis: A Practical Experience Report." In: *Dependable Systems and Networks*. 2016, pp. 662–669. DOI: 10.1109/DSN.2016.67.

[142] David M. Russinoff. "A Mechanically Checked Proof of IEEE Compliance of the Floating Point Multiplication, Division and Square Root Algorithms of the AMD-K7™ Processor." In: *LMS Journal of Computation and Mathematics* 1 (1998), pp. 148–200. DOI: 10.1112/S1461157000000176.

[143] Maximilian Schäffeler. "Verified Solution Methods for Markov Decision Processes." MA thesis. Technische Universität München, 2021. 56 pp.

[144] Maximilian Schäffeler and Mohammad Abdulaziz. "Formally Verified Approximate Policy Iteration." In: *AAAI Conference on Artificial Intelligence*. Vol. 39. AAAI Press, 2025. To appear.

[145] Maximilian Schäffeler and Mohammad Abdulaziz. "Formally Verified Solution Methods for Markov Decision Processes." In: *AAAI Conference on Artificial Intelligence*. Vol. 37. 12. AAAI Press, 2023, pp. 15073–15081. DOI: 10.1609/aaai.v37i12.26759.

[146] Daniel Selsam, Percy Liang, and David L. Dill. "Developing Bug-Free Machine Learning Systems With Formal Mathematics." In: *International Conference on Machine Learning*. Vol. 70. Proceedings of Machine Learning Research. PMLR, 2017, pp. 3047–3056.

[147] Alexey Solovyev, Marek S. Baranowski, Ian Briggs, Charles Jacobsen, Zvonimir Rakamarić, and Ganesh Gopalakrishnan. "Rigorous Estimation of Floating-Point Round-Off Errors with Symbolic Taylor Expansions." In: *ACM Trans. Program. Lang. Syst.* 41.1 (2018), 2:1–2:39. DOI: 10.1145/3230733.

[148] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, 2018. 552 pp.

[149] Yong Kiam Tan, Jiong Yang, Mate Soos, Magnus O. Myreen, and Kuldeep S. Meel. "Formally Certified Approximate Model Counting." In: *Computer Aided Verification*. Springer, 2024, pp. 153–177. DOI: 10.1007/978-3-031-65627-9_8.

[150] Joseph Tassarotti and Robert Harper. "Verified Tail Bounds for Randomized Programs." In: *Interactive Theorem Proving*. Springer, 2018, pp. 560–578. DOI: 10.1007/978-3-319-94821-8_33.

[151] René Thiemann and Akihisa Yamada. "Formalizing Jordan Normal Forms in Isabelle/HOL." In: *Certified Programs and Proofs*. Association for Computing Machinery, 2016, pp. 88–99. DOI: 10.1145/2854065.2854073.

[152] Laura Titolo, Mariano Moscato, Marco A. Feliu, Paolo Masci, and César A. Muñoz. "Rigorous Floating-Point Round-Off Error Analysis in PRECiSA 4.0." In: *Formal Methods*. Springer, 2025, pp. 20–38. DOI: 10.1007/978-3-031-71177-0_2.

[153] Laura Titolo, Mariano M. Moscato, César A. Muñoz, Aaron Dutle, and François Bobot. "A Formally Verified Floating-Point Implementation of the Compact Position Reporting Algorithm." In: *Formal Methods*. Springer, 2018, pp. 364–381. DOI: 10.1007/978-3-319-95582-7_22.

[154] Rishikesh Hirendu Vaishnav. "Formalizing the Beginnings of Bayesian Probability Theory in the Lean Theorem Prover." UC San Diego, 2022.

[155] Koundinya Vajjha, Avraham Shinnar, Barry M. Trager, Vasily Pestun, and Nathan Fulton. "CertRL: Formalizing Convergence Proofs for Value and Policy Iteration in Coq." In: *Certified Programs and Proofs*. ACM, 2021, pp. 18–31. DOI: 10.1145/3437992.3439927.

[156] Koundinya Vajjha, Barry M. Trager, Avraham Shinnar, and Vasily Pestun. "Formalization of a Stochastic Approximation Theorem." In: *Interactive Theorem Proving*. Vol. 237. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 31:1–31:18.

[157] Eelis van der Weegen and James McKinna. "A Machine-Checked Proof of the Average-Case Complexity of Quicksort in Coq." In: *Types for Proofs and Programs*. Springer, 2009, pp. 256–271. DOI: 10.1007/978-3-642-02444-3_16.

[158] Floris van Doorn. "Formalized Haar Measure." In: *Interactive Theorem Proving*. Ed. by Liron Cohen and Cezary Kaliszyk. Vol. 193. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, 18:1–18:17. DOI: 10.4230/LIPIcs.ITP.2021.18.

[159] D. J. White. "A Survey of Applications of Markov Decision Processes." In: *Journal of the Operational Research Society* 44.11 (1993), pp. 1073–1096. DOI: 10.1057/jors.1993.181.

[160] Ralf Wimmer, Nils Jansen, Erika Ábrahám, Joost-Pieter Katoen, and Bernd Becker. "Minimal Counterexamples for Linear-Time Probabilistic Verification." In: *Theoretical Computer Science* 549 (2014), pp. 61–100. DOI: 10.1016/j.tcs.2014.06.020.

[161] Haoze Wu, Omri Isac, Aleksandar Zeljic, Teruhiro Tagomori, Matthew L. Daggitt, Wen Kokke, Idan Refaeli, Guy Amir, Kyle Julian, Shahaf Bassan, Pei Huang, Ori Lahav, Min Wu, Min Zhang, Ekaterina Komendantskaya, Guy Katz, and Clark W. Barrett. "Marabou 2.0: A Versatile Formal Analyzer of Neural Networks." In: *Computer Aided Verification*. Vol. 14682. Lecture Notes in Computer Science. Springer, 2024, pp. 249–264. DOI: 10.1007/978-3-031-65630-9_13.

[162] Yuhuai Wu, Albert Q. Jiang, Wenda Li, Markus N. Rabe, Charles Staats, Mateja Jamnik, and Christian Szegedy. "Autoformalization with Large Language Models." In: *International Conference on Neural Information Processing Systems*. NIPS '22. Curran Associates Inc., 2022, pp. 32353–32368.

[163] Kexing Ying and Rémy Degenne. "A Formalization of Doob's Martingale Convergence Theorems in Mathlib." In: *Certified Programs and Proofs*. Association for Computing Machinery, 2023, pp. 334–347. DOI: 10.1145/3573105.3575675.

[164] Håkan LS Younes and Michael L. Littman. "PPDDL1. 0: The Language for the Probabilistic Part of IPC-4." In: *Proc. International Planning Competition*. 2004, pp. 70–74.

[165] Emily Yu, Armin Biere, and Keijo Heljanko. "Progress in Certifying Hardware Model Checking Results." In: *Computer Aided Verification*. Springer, 2021, pp. 363–386. DOI: 10.1007/978-3-030-81688-9_17.

[166] Emily Yu, Nils Froleyks, Armin Biere, and Keijo Heljanko. "Towards Compositional Hardware Model Checking Certification." In: *Formal Methods in Computer Aided Design*. IEEE, 2023, pp. 1–11. DOI: 10.34727/2023/isbn.978-3-85448-060-0_12.

[167] Lei Yu. "A Formal Model of IEEE Floating Point Arithmetic." In: *Archive of Formal Proofs* (2013). URL: https://isa-afp.org/entries/IEEE_Floating_Point.html.

Appendix

This appendix includes the original versions of the publications that are part of the cumulative dissertation. For each publication, we provide the original manuscript, as well as the publisher's permission to include it in this dissertation.

Core Publication 1 (Factor of core: 1, sole first author)
Maximilian Schäffeler and Mohammad Abdulaziz. "Formally Verified Solution Methods for Markov Decision Processes." In: *AAAI Conference on Artificial Intelligence*. Vol. 37. 12. AAAI Press, 2023, pp. 15073–15081. DOI: 10.1609/aaai.v37i12.26759

Core Publication 2 (Factor of core: 1, sole first author)
Maximilian Schäffeler and Mohammad Abdulaziz. "Formally Verified Approximate Policy Iteration." In: *AAAI Conference on Artificial Intelligence*. Vol. 39. AAAI Press, 2025. To appear.

Core Publication 3 (Factor of core: 0.5, two first authors)
Bram Kohlen, Maximilian Schäffeler, Mohammad Abdulaziz, Arnd Hartmanns, and Peter Lammich. "A Formally Verified IEEE 754 Floating-Point Implementation of Interval Iteration for MDPs." In: *Computer Aided Verification*. Lecture Notes in Computer Science. Springer, 2025. To appear.

Core Publication 4 (Factor of core: 0.17, six first authors)
Krishnendu Chatterjee, Tim Quatmann, Maximilian Schäffeler, Maximilian Weininger, Tobias Winkler, and Daniel Zilken. "Fixed Point Certificates for Reachability and Expected Rewards in MDPs." In: *Tools and Algorithms for the Construction and Analysis of Systems*. Lecture Notes in Computer Science. Springer, 2025. To appear.

Core Publication 1

Maximilian Schäffeler and Mohammad Abdulaziz. "Formally Verified Solution Methods for Markov Decision Processes." In: *AAAI Conference on Artificial Intelligence*. Vol. 37. 12. AAAI Press, 2023, pp. 15073–15081. DOI: 10.1609/aaai.v37i12.26759

The Thirty-Seventh AAAI Conference on Artificial Intelligence (AAAI-23)

Formally Verified Solution Methods for Markov Decision Processes

Maximilian Schäffeler[1], **Mohammad Abdulaziz**[1,2]

[1]Technische Universität München, Germany
[2]King's College London, United Kingdom
maximilian.schaeffeler@tum.de, mohammad.abdulaziz@kcl.ac.uk

Abstract

We formally verify executable algorithms for solving Markov decision processes (MDPs) in the interactive theorem prover Isabelle/HOL. We build on existing formalizations of probability theory to analyze the expected total reward criterion on finite and infinite-horizon problems. Our developments formalize the Bellman equation and give conditions under which optimal policies exist. Based on this analysis, we verify dynamic programming algorithms to solve tabular MDPs. We evaluate the formally verified implementations experimentally on standard problems, compare them with state-of-the-art systems, and show that they are practical.

Introduction

Despite the impressive advances in the capabilities of different types of AI systems, it is becoming clear that one major hurdle to their wide adoption is the lack of trustworthiness of these systems. This has prompted researchers to study techniques to boost the trustworthiness of AI systems in different areas, like machine learning (Selsam, Liang, and Dill 2017; Katz et al. 2017), planning (Abdulaziz, Gretton, and Norrish 2019; Abdulaziz and Lammich 2018), and model-checking (Esparza et al. 2013). However, one of the areas of AI where there is still a lot to be done regarding trustworthiness is software for solving Markov decision processes (MDPs). MDPs are models for systems where a decision-maker selects actions with random outcomes to maximize long-term rewards. Such systems with uncertainty occur in wide-ranging areas, e.g. planning, reinforcement learning, model checking, and operations research. Depending on the area, the questions one asks about the model might be different, e.g. in planning the aim might be to find a policy that chooses an action for a robot in every state, with some optimality guarantees on the accrued rewards, while in model-checking the aim might be, for instance, to find what the expected delays of a real-time system are. Furthermore, a lot of safety-critical systems could be modeled as MDPs, e.g. a policy (aka strategy) could be used to navigate an autonomous vehicle. Like with other AI systems, for such applications, it is strongly desirable to have trustworthy programs to reason about and solve MDPs.

A methodology with notable success in developing provably correct software involves the use of interactive theorem provers (ITPs). Success stories of the use of ITPs to develop trustworthy software include a formally verified OS kernel (Klein et al. 2009), a verified compiler for C (Leroy 2009), and verified implementations of a multitude of algorithms (Nipkow, Eberl, and Haslbeck 2020).

In this work, we study the application of ITPs to the development of trustworthy software for reasoning about and solving MDPs using iterative methods. However, using ITPs to develop verified implementations of these iterative methods brings its own particular set of challenges, compared to developing other types of verified algorithms. First, at a mathematical level, formal proofs of correctness of MDP solving algorithms in ITPs need a combination of diverse and significantly non-trivial formal mathematical libraries and concepts, e.g. probabilities, limits, (co)recursion, and probabilistic transition systems. Second, at an implementation level, multiple challenges exist, e.g. should the algorithm be implemented imperatively or functionally, what kind of implementation of numerics should be used, etc. Last, an even bigger challenge is the overall architecture of the verified system: should it be a verified implementation of an iterative method, or should we use an unverified system to perform the computation and produce a certificate, which is later validated using a formally verified certificate checker.

In addressing those challenges, multiple factors play in, like the feasibility of the verification, the reusability of the results, and the efficiency or practicality of the verified implementation. Although previous authors (Vajjha et al. 2021; Chevallier and Fleuriot 2021) have addressed the problem of verifying iterative methods for MDPs using ITPs, all of them focused on the abstract mathematical challenges.

In this paper, we comprehensively tackle all three challenges. At the abstract mathematical level, using the ITP Isabelle/HOL (Nipkow, Paulson, and Wenzel 2002) and following the exposition of Puterman (Puterman 1994), we develop a general formalization of MDPs with rewards. Based on that, we first formally prove correct the backward induction algorithm for finite-horizon MDPs. Second, for infinite-horizon problems, we model four fundamental iteration-based methods in Isabelle/HOL, namely, value iteration, policy iteration, modified policy iteration, and splitting-based methods. We prove the correctness of all methods against

a formal specification of expected total discounted reward. Compared to previous attempts (Vajjha et al. 2021; Chevallier and Fleuriot 2021), at a mathematical level, our formalization is more reusable. We also show the correctness of the algorithms against a simpler definition of expected total discounted reward.

As our second contribution, we devise the first executable verified implementations of the four algorithms of which we are aware. In the process, we fix a mistake in a textbook correctness proof of Gauss-Seidel value iteration. We experimentally evaluate our implementations of the four algorithms on standard probabilistic planning problems and show that they are practical. Our implementations use efficient data structures and can solve planning problems with millions of transitions.

Finally, we experimentally show that combining our verified infinite-horizon implementations with an unverified implementation yields significant performance improvements. In particular, we show that one can use a fast floating-point implementation to perform all the iterations and then use the formally verified implementation for the last iteration, effectively having the best of both worlds. The formalization, our verified implementation, and detailed benchmarks are available online.[1]

Background

Isabelle/HOL

An ITP is a program that implements a formal mathematical system, in which definitions and theorem statements are written, and a set of axioms or derivation rules, using which proofs are constructed. To prove a fact in an ITP, the user provides high-level steps of a proof, and the ITP fills in the details, at the level of axioms, culminating in a formal proof. We performed the formalization in this paper using the interactive theorem prover Isabelle/HOL (Nipkow, Paulson, and Wenzel 2002), a theorem prover for Higher-Order Logic. Roughly speaking, Higher-Order Logic can be seen as a combination of functional programming with logic. Isabelle/HOL supports the extraction of the functional fragment to executable code (Haftmann and Nipkow 2007).

Isabelle is designed for trustworthiness: following the Logic for Computable Functions (LCF) approach (Milner 1972), a small kernel implements the inference rules of the logic, and, using encapsulation features of abstract data types, it guarantees that all theorems are actually proved by this small kernel. Around the kernel, there is a large set of tools that implement proof tactics and high-level concepts like algebraic data types and recursive functions. Bugs in these tools cannot lead to invalid proofs, but only to error messages when the kernel refuses a proof.

Probability Theory

Probability theory was formalized in Isabelle/HOL primarily by Hölzl based on a library of measure theory (Hölzl 2013). Let Ω be the sample space of the probability space P. We denote the expectation of a random variable $X : \Omega \to \mathbb{R}$ by $\mathbb{E}_{\omega \sim P}[X(\omega)]$, it is defined via the Lebesgue integral. We write $\mathcal{P}(\Omega)$ for the set of all probability spaces over Ω.

[1] https://github.com/schaeffm/mdps-isabelle-hol

Bounded Functions

Our formal analysis of MDPs makes use of bounded functions. For a set X and a complete normed vector space V, the space of bounded functions $X \to_b V$ contains all functions where the image of X is bounded w.r.t. the norm on V. We define a pointwise addition and scaling operation, and a partial order for bounded functions. The uniform norm makes bounded functions a complete normed vector space.

The set of bounded linear functions $V \to_L W$ between normed vector spaces consists of all linear transformations that perform bounded scaling. For finite-dimensional vector spaces, bounded linear functions correspond to matrices. Equipped with the operator norm $\|A\| := \sup_{v \in V} \|Av\|/\|v\|$, bounded linear functions also form a normed vector space. We formalize the following closed-form formula for the geometric series of contractive bounded linear functions, that is central to our analysis:

Theorem 1 (Puterman 1994, Corollary C.4)**.** *Let $A : V \to_L W$ with $\|A\| < 1$, then $\sum_{i \in \mathbb{N}} A^i = (1 - A)^{-1}$.*

Bounded linear functions are available in the Isabelle/HOL distribution as a subtype of the function type. A type for bounded continuous functions is also present in the library, which we generalize to bounded functions. Introducing special function types has the advantage that we can then show that they are instances of several vector space type classes (Haftmann and Wenzel 2006) and profit from an existing library of theorems and overloaded notation for vector spaces.

Markov Decision Processes

In this section we give an overview of our formalization of the mathematical concepts needed for specifying the different algorithms on MDPs and their correctness criteria. We closely follow the exposition in (Puterman 1994, Chapters 2-6). However, the most notable outcome of our formalization is that we give a more explicit construction of the trace space of the MDP and we also correct mistakes in the proofs.

Markov decision processes (Bellman 1957) model systems where an agent acts in an environment that exhibits randomized behavior. The dynamics of MDPs evolve over a succession of epochs, during each of which the agent analyzes the current state of the environment, chooses an action, obtains a reward, and transitions to a new state. The goal of the agent is to optimize the action selection to maximize the accrued rewards. We distinguish between finite and infinite horizon problems, where the number of epochs is finite or infinite respectively. Our formalization defines MDPs on general state and action spaces, but our analysis of solution methods only covers discrete state and action spaces.

Definition 1 (Discrete Markov Decision Process with Rewards)**.** *A Markov decision process is composed of discrete sets of states S and actions A, a Markov kernel of transition probabilities $K : S \times A \to \mathcal{P}(S)$, a bounded reward function $r : S \times A \to_b \mathbb{R}$, and state rewards $r_N : S \to_b \mathbb{R}$. For*

each state-action pair, r gives a real-valued reward, while r_N denotes the final value of a state (for finite-horizon problems). Furthermore, we assume the existence of a non-empty set of enabled actions $A_s \subseteq A$ for each state $s \in S$.

We use *locales* (Ballarin 2014) to define MDPs in Isabelle/HOL. A locale can be seen as a formal mathematical context that introduces constants and assumptions, in which we develop our formalization. Locales can be instantiated, e.g. with a concrete MDP. This requires a proof that discharges the assumptions, yielding all the theorems proved within the locale. There already exists a formalization of MDPs in the Archive of Formal Proofs (Hölzl 2017). However, this formalization does not support stochastic action choice, an important component of our analysis. Hence we develop a more flexible definition of MDPs ourselves.

Construction of the Trace Space

In each epoch, the agent determines the action with a decision rule $d : S \to \mathcal{P}(A)$, i.e. it chooses a probability distribution over the enabled actions. A deterministic decision rule is a function $d : S \to A$ that respects enabled actions. In general, the decision rule employed by the agent may depend upon the history of all previous decisions and states visited. Histories are alternating sequences of states and actions, they form the set H. A policy $\pi : H \times S \to \mathcal{P}(A)$ can be seen as a mapping from histories to decision rules. We denote the set of all decision rules by D and the set of policies by Π. Markovian policies depend only on the current epoch and may thus be represented as a sequence of decision rules. Stationary policies are policies that only use a single decision rule. In our formalization, each subclass of policies has a different type. We insert explicit coercion functions to translate between different types of policies, but we omit them here for notational economy.

We now fix a policy π and an initial state. That determines the trace space of the MDP, the probability space of infinite sequences of state-action pairs as observed by the agent. The law of this stochastic process is $\kappa : H \to \mathcal{P}(S \times A)$:

$$\kappa(s) \quad := \textbf{do}\{\ a \leftarrow \pi(s);\ \text{return}\ (s,a)\ \} \tag{1}$$

$$\kappa(h,s,a) := \textbf{do}\{\ s' \leftarrow K(s,a);\ a' \leftarrow \pi(h,s,a,s');\ \text{return}\ (s',a')\ \}. \tag{2}$$

The definition uses the Giry monad (Giry 1982) on probability spaces to elegantly compose a sequence of experiments. A more detailed introduction to the Giry monad and **do**-notation is given in the appendix. Specifically, the law κ shows how to extend a finite trace by a single state-action pair: we sample the next state s' using the Markov kernel K on the current state-action pair from the history, then choose an action, and finally return the next state-action pair. From the law of the stochastic process, we obtain the trace space $\mathcal{T}^\pi : S \to \mathcal{P}((S \times A)^\infty)$ via the Ionescu-Tulcea extension theorem. The theorem was formalized by Hölzl, who used it to construct trace spaces for Markov chains (Hölzl 2017).

Alternatively, the stochastic process can be viewed as a sequence of state-action distributions, one for each epoch. The state-action distribution $P^n_\pi : S \to \mathcal{P}(S \times A)$ at epoch n is obtained as a projection from $\mathcal{T}^\pi$. We prove that P^n_π adheres to the following recursive characterization:

$$P^0_\pi(s) \quad = \textbf{do}\{\ a \leftarrow \pi(s);\ \text{return}\ (s,a)\ \} \tag{3}$$

$$P^{n+1}_\pi(s) = \textbf{do}\{\ a \leftarrow \pi(s);\ s' \leftarrow K(s,a);\ P^n_{\pi'}(s')\ \} \tag{4}$$

$$\text{where}\ \pi'((s_1,a_1),\ldots,t) = \pi((s,a),(s_1,a_1),\ldots,t).$$

From these equations we derive that for a fixed initial state $s \in S$ and an arbitrary policy π, there exists a Markovian policy π_M such that $P^n_\pi(s) = P^n_{\pi_M}(s)$ for all n. This fact allows us to restrict the search for optimal infinite-horizon policies to Markovian policies.

Finite-Horizon MDPs

For a finite horizon N, we denote the value of a policy π as $\nu^\pi_N : S \to_b \mathbb{R}$. It is defined as the expected discounted sum of rewards accrued over time, plus a final state reward:

$$\nu^\pi_N(s) := \mathbb{E}_{\omega \sim \mathcal{T}^\pi(s)}\left[\textstyle\sum_{i<N} \lambda^i r(\omega_i) + r_N(\omega_{N,1})\right]. \tag{5}$$

Typically for finite-horizon problems the discount factor $\lambda = 1$. We show that the optimal finite-horizon value $\nu^*_N := \sup_{\pi \in \Pi} \nu^\pi_N$ can be computed using backward induction as $\nu^*_N = u^*_0$, where $u^*_N(s) := r_N(s)$ and for $t < N$

$$u^*_t(s) := \sup_{a \in A_s} r(s,a) + \lambda \mathbb{E}_{K(s,a)}\left[u^*_{t+1}\right].$$

The maximizing actions in each equation constitute an optimal, Markovian, and deterministic policy.

Infinite-Horizon MDPs

We also define the infinite-horizon value $\nu^\pi : S \to_b \mathbb{R}$

$$\nu^\pi(s) := \lim_{n \to \infty} \nu^\pi_n(s). \tag{6}$$

For infinite-horizon problems we assume $0 \leq \lambda < 1$, as well as $r_N = 0$. The discount factor decreases the relevance of later rewards, and also guarantees that each policy has a finite value. We show that ν^π is a member of the space V_B of bounded functions $S \to_b \mathbb{R}$. Here, we would like to note that both existing formalizations of the expected total discounted reward criterion (Chevallier and Fleuriot 2021; Vajjha et al. 2021) avoid the construction of the trace space and, instead, use the following characterization as their definition of ν^π:

$$\nu^\pi(s) = \textstyle\sum_{i \in \mathbb{N}} \lambda^i\, \mathbb{E}_{P^i_\pi(s)}[r]. \tag{7}$$

We argue that our definition is more natural and simpler, as it is derived from the trace space of the MDP, and is the one used in the textbook exposition, e.g. Puterman 1994, Equation 5.1.3. Thus, we believe it is a superior basis for the verification of specifications in terms of ν^π. However, the definition of ν^π is difficult to work with directly, because reasoning w.r.t. the trace space is complex. We therefore show that our definition and (7) are equivalent.

We further simplify ν^π with the introduction of a vector notation. The reward vector for a decision rule d is defined as $r^d_s := \mathbb{E}_{a \sim d(s)}[r(s,a)]$. We also define the operator $\mathcal{P}^n_\pi : V_B \to_L V_B$, a bounded linear function that is equivalent to the n-step transition probability matrix for policy π. $\mathcal{P}^n_\pi$ acts in the same way on a function as a stochastic matrix does on a vector. We prefer statements in vector notation in our

formalization, so we can prove theorems at a high level of abstraction. Now we can formulate ν^π in vector notation for Markovian policies π:

$$\nu^\pi = \sum_i \lambda^i \mathcal{P}_\pi^i r^{\pi_i},\ (\mathcal{P}_\pi^i v)_s \coloneqq \mathbb{E}_{(s',a')\sim P_\pi^i(s)}\left[v_{s'}\right]. \quad (8)$$

Ultimately our goal is to maximize the expected total discounted reward by optimizing the policy. We define the value of the MDP $\nu^* \coloneqq \sup_{\pi\in\Pi}\ \nu^\pi$ to be the least upper bound of ν^π ranging over all policies π. As noted earlier, it suffices to consider Markovian policies for optimality.

Policy Evaluation Before searching for an optimal policy, we first tackle the problem of policy evaluation, i.e. the computation of ν^π. Specifically, we only consider a decision rule d. It is possible to compute ν^d exactly, as from Theorem 1 we can show that $\nu^d = (1 - \lambda\mathcal{P}_d)^{-1} r^d$. However, determining an exact solution becomes computationally expensive for large state spaces. A more practical alternative is the approximation using a fixed-point iteration. We define the Bellman operator $L_d(v) \coloneqq r^d + \lambda P_d v$ and observe that $\nu^d = L_d(\nu^d)$. Since L_d is a contraction mapping, the Banach fixed-point theorem establishes that ν^d is its unique fixed-point and that $\lim_{i\to\infty} L_d^i(v) = \nu^d$ for any $v \in V_B$.

Optimality Equations The Bellman operator can be used to approximate ν^*. We therefore introduce the Bellman optimality operator $\mathcal{L} : V_B \to V_B$ where

$$\mathcal{L}(v) \coloneqq \sup_{d\in D}\ L_d(v). \quad (9)$$

The proof that $\mathcal{L}$ is well-defined is missing from Puterman's book, as discussed by Chevallier and Fleuriot 2021. We can prove that the supremum is indeed well-defined since changing the action selection for one state does not influence the value of other states. Next, we prove that every fixed-point of $\mathcal{L}$ is equal to the optimal value ν^*. Since $\mathcal{L}$ is also a contraction mapping, ν^* is actually the unique fixed-point of $\mathcal{L}$ and can be computed with a fixed-point iteration.

Theorem 2 (Fixed point of $\mathcal{L}$)**.** *The optimal value ν^* is the unique fixed point of the optimality operator $\mathcal{L}$. Additionally, $\lim_{i\to\infty} \mathcal{L}^i(v) = \nu^*$ for $v \in V_B$.*

It remains to be shown under which conditions an optimal policy (one that achieves the value of the MDP) exists. A sufficient condition is the existence of the supremum in the definition of $\mathcal{L}(v)$ for every $v \in V_B$. The optimal policy is then the decision rule d^* that maximizes $L_{d^*}(\nu^*)$. If the set of enabled actions in each state is finite, the supremum in $\mathcal{L}$ is always attained. Thus finite MDPs have optimal policies that are stationary and deterministic. Finally, we show that if there exists an optimal policy, there also exists an optimal stationary and deterministic policy.

Infinite-Horizon Algorithms

A novelty of our work is that we verify optimized and executable algorithms to compute optimal policies. Verifying such executable algorithms pertains to two tasks. The first is formalizing the algorithm in the logic of the theorem prover, usually as a non-executable recursive function. This function should capture the mathematical essence of the algorithm, but excludes implementation details. Then one proves the desired mathematical properties of this function, most notably termination and that the algorithms compute optimal policies. In the second step, we formally verify executable versions of the algorithms at hand. We now discuss the first step and then later discuss the second.

From here on, we assume that both S and A are finite. This ensures the existence of optimal policies and allows us to extract executable code. We shortly present the basic value iteration and policy iteration algorithms and then give a more detailed exposition of the optimized variants, namely modified policy iteration and Gauss-Seidel value iteration. For finite-horizon MDPs, we compute optimal policies using the backward induction algorithm described earlier.

The value iteration algorithm is based on Theorem 2. It repeatedly applies the Bellman optimality operator $\mathcal{L}$ to an initial estimate of the value function and stops as soon as successive iterates are sufficiently close in distance to achieve the desired accuracy. On the other hand, the policy iteration algorithm performs a direct search in the space of deterministic decision rules (Puterman 1994, Section 6.4). It alternates between evaluating a candidate policy and improving it, until the policy stabilizes. Exact policy evaluation involves solving a system of linear equations, for which we use a verified implementation of the Gauss-Jordan algorithm (Thiemann and Yamada 2016).

Modified Policy Iteration Value iteration and policy iteration can be considered extreme instances of modified policy iteration (Algorithm 1) (Puterman 1994, Section 6.5). In each step, value iteration only approximates the value of the current policy with a single iteration of the Bellman operator, while policy iteration considers infinitely many iterations. Modified policy iteration places a varying amount of policy evaluation steps between each policy improvement phase. When the initial value estimate v is conservative, i.e. $v \leq \mathcal{L}(v)$, the algorithm terminates with a policy that is optimal upto an a priori error bound (ϵ-optimal policy). The convergence proof of modified policy iteration is based on the observation that its iterates dominate the iterates of value iteration but never exceed ν^*.

Gauss-Seidel Value Iteration Finally, Gauss-Seidel value iteration (Algorithm 2) is a variant of value iteration where the value estimates are updated in-place (Puterman 1994, Section 6.3.3). The algorithm delivers an ϵ-optimal policy. It converges at least as fast as value iteration, because updated estimates are used immediately, not only in the next iteration. In practice, often substantially fewer iterations are required until convergence. We now assume that the state space of the MDP is a subset $\{0, 1, \ldots, n\}$ of the natural numbers and each iteration proceeds from low to high states. Thus we can interpret bounded linear functions as matrices.

Puterman shows that Gauss-Seidel value iteration is an instance of a class of algorithms called splitting methods. These use regular splittings (Q_d, R_d) of the linear function $(1 - \lambda P_d) = Q_d - R_d$. Regularity of a splitting means that Q_d^{-1} and R_d are nonnegative matrices. The algorithm proceeds the same way as value iteration with the Bellman operator replaced by $G_d(v) \coloneqq Q_d^{-1}(r^d + R_d v)$. For the Gauss-Seidel method we split $P_d = P_d^L + P_d^U$ into a

strictly lower triangular part P_d^L and an upper triangular part P_d^U and choose the regular splitting $Q_d = (1 - \lambda P_d^L)$ and $R_d = \lambda P_d^U$. In-place value iteration should follow the equation $v^{n+1} = r^d + \lambda P_d^L v^{n+1} + \lambda P_d^U v^n$ for some d. Rearranging terms then directly yields

$$v^{n+1} = (1 - \lambda P_d^L)^{-1}(r^d + \lambda P_d^U v^n) = G_d(v^n). \quad (10)$$

We define the Gauss-Seidel variant of the Bellman optimality operator $\mathcal{G}(v) \coloneqq \sup_{d \in D} G_d(v)$. We show that for any regular splitting, $\mathcal{G}$ is a contraction mapping and the algorithm converges if the supremum in $\mathcal{G}(v)$ is attained for all $v \in V_B$ and $\sup_{d \in D} \|Q_d^{-1} R_d\| < 1$. The requirement that the supremum has to be attained is overlooked by Puterman, where the existence of such a decision rule is implicitly assumed. We close this proof gap for Gauss-Seidel value iteration. Assuming a total ordering on the states, as part of our formalization, we show that a decision rule that attains the supremum for all states smaller than s can always be extended to an optimal decision rule up to and including state s. Before we state the theorem, let $f(x := y)$ denote the pointwise update of the value of function f at x.

Theorem 3. *Assume that:*

1. *the decision rule d^* maximizes $G_{d^*}(v)$ for all states smaller than s, i.e. $(G_d(v))_t \leq (G_{d^*}(v))_t$, for any $t < s$ and $d \in D_D$, and*
2. *the maximizing action in s is a, i.e. for any $a' \in A_s$, $(G_{d^*(s:=a')}(v))_s \leq (G_{d^*(s:=a)}(v))_s$.*

Then $d^*(s := a)$ maximizes $G_{d^*}(v)$ for all states up to s, i.e. $(G_d(v))_t \leq (G_{d^*(s:=a)}(v))_t$, for any $t \leq s$ and $d \in D_D$.

Proof. We obtain $G_d(v) = r^d + \lambda P_d^U v + \lambda P_d^L G_d(v)$ by rearranging the definition of G_d. Since P_d^L is lower diagonal, the value of $(G_d(v))_t$ depends only on the values of d up to state t. Now, it follows from the assumptions that $d^*(s := a)$ is already maximizing for all $t < s$. Thus it suffices to show that for all deterministic decision rules d, $(G_d(v))_s \leq (G_{d^*(s:=a)}(v))_s$. With $d' \coloneqq d^*(s := d(s))$, we have

$$\begin{aligned}
&(P_d^L\ G_d(v))_s && (11)\\
&= (P_{d'}^L G_d(v))_s && P^L \text{ only depends on } d \text{ at } s \quad (12)\\
&\leq (P_{d'}^L G_{d^*}(v))_s && d^* \text{ optimal, } P^L \text{ is triangular} \quad (13)\\
&\leq (P_{d'}^L G_{d'}(v))_s. && G_{d^*} \text{ independent of } d^* \text{ above } s \quad (14)
\end{aligned}$$

So $(G_d(v))_s = (r^d + \lambda P_d^U v + \lambda P_d^L G_d(v))_s \leq (G_{d'}(v))_s$. Finally assumption 2 lets us complete the proof. □

In Puterman's book, a proof of the ϵ-optimality of this splitting-based method is not given, which we supplement. However, that requires us to change the last step of the algorithm from how it was presented originally (Puterman 1994, Section 6.3.3). There, the policy is determined using a basic value iteration step. To prove ϵ-optimality, we need to, instead, use a Gauss-Seidel step to determine the policy.

Verifying an Executable Implementation

Our approach to obtaining executable versions of the algorithms is based on step-wise refinement (Wirth 1971). In this approach, an executable specification of an algorithm is derived from an unexecutable one by replacing mathematical structures by data structures. We make heavy use of locales in our refinement. The built-in data-refinement of the executable code generator of Isabelle/HOL proved to be too inflexible, as it does not allow the same mathematical structure to be implemented with different data structures at different places in the algorithm.

Initially, we have abstract definitions of the algorithms as presented above. In the first step, we show that the reward function and the transition system can be represented as finite maps, and that action sets can be represented as finite sets. The interfaces of data structures implementing finite sets and maps are available as locales in the Isabelle/HOL distribution. These locales come with interpretations for concrete data structures based on e.g. balanced trees or hash maps. We prove correctness on the level of the abstract interfaces, which gives us the flexibility to choose between several concrete data structures without modifying proofs. In our final version, we represent an MDP as an array with an entry for each state. The array stores red-black trees that map actions to rewards and transitions.

We use the code generation facilities provided by Isabelle/HOL (Haftmann and Nipkow 2007) to extract executable Standard ML code from the formally verified algorithms. A wrapper parses problems from a file and passes them to the verified algorithms. To obtain numerically accurate results, real numbers are represented as rational numbers. This representation comes with an ever greater performance penalty as the number of iterations increases and the fractions become larger. For comparison, we also provide a separate implementation using floating-point arithmetic. However, due to the possibility of floating-point errors accumulating, we lose the formal guarantees.

Algorithm 1: Modified Policy Iteration

Input : $v \in V_B$ where $v \leq \mathcal{L}(v)$, $m : \mathbb{N} \to \mathbb{N}$
for $i \in 0 \ldots$ **do**
  for $s \in S$ **do** $d(s) \leftarrow \arg\max_{a \in A_s}(L_a(v))_s$
  if $d_\infty(v, \mathcal{L}(v)) < \frac{\epsilon(1-\lambda)}{2\lambda}$ **then return** d
  $v \leftarrow L_d^{m_i+1}(v)$

Algorithm 2: Gauss-Seidel Value Iteration

Input : $v \in V_B$
repeat
  $v_{old} \leftarrow v$
  for $s \in S$ **do** $v(s) \leftarrow \max_{a \in A_s}(L_a(v))_s$
until $d_\infty(v, v_{old}) < \frac{\epsilon(1-\lambda)}{2\lambda}$
for $s \in S$ **do**
  $d(s) \leftarrow \arg\max_{a \in A_s}(L_a(v))_s$
  $v(s) \leftarrow \max_{a \in A_s}(L_a(v))_s$
return d

Notes on the Formalization

Since a main goal of this project is to showcase theorem proving as a methodology to obtain verified algorithms for

probabilistic systems, we note some of the lessons we learnt and the challenges encountered during the formalization. This section should be of particular interest to readers who would be interested in verifying similar algorithms in Isabelle/HOL or in other theorem provers.

Our work builds on formalization efforts in probability theory (Hölzl 2013) and linear algebra from the archive of formal proofs (AFP) and the Isabelle distribution. The construction of MDPs and their trace space extends previous work in Isabelle/HOL on Markov Chains and MDPs (Hölzl 2017). This library of formalized mathematics was crucial to make our verification project viable.

Isabelle/HOL provides us with powerful tools for automated reasoning, such as the proof method *auto*, and access to external automated theorem provers via *sledgehammer* (Paulson and Blanchette 2010). The strong automation combined with the structured proof language Isar (Wenzel 1999) enables the development of maintainable, reusable and human-readable proofs. However, in our formalization we encounter situations where automation breaks. For instance, chains of equations involving mathematical operators that are potentially undefined, like infinite sums or integrals require duplicate work. We first need to show that the operation maintains the desired property, e.g. summability, measurability, boundedness and integrability. Only then can we show that the algebraic manipulation is actually correct. This problem becomes especially apparent with nested sums or integrals. A potential, albeit still preliminary, solution to this problem has been proposed by (Coen and Zoli 2008).

To improve the reusability of our work, we formalize MDPs with arbitrary, uncountable state spaces and probabilistic action choice. Still, the correctness of the verified algorithms only holds for finite-state MDPs. Our initial formalization of the algorithms models the state space of the MDP as a type, i.e. all the states have to be known at compile-time. Initially, our intention was to use the Isabelle tool *types-to-sets* (Kuncar and Popescu 2016) to generalise our formalization to be parameterised by the set of states of the MDP, and get algorithms operating on the corresponding sets of states. Nonetheless, we could not transfer statements involving arbitrary probability spaces automatically using that tool, as its automation is also still preliminary.

Experimental Evaluation

We evaluate our algorithms on a set of standard benchmarks and compare the results to a reference implementation. Furthermore, we show how our implementation can be supported with solutions obtained by unverified solvers.

Setup We benchmark our implementation on explicitly represented MDPs, which are compiled problems from the *International Planning Competition 2018*.[2] Most domains come with multiple instances of different sizes. The problems are parsed by an unverified wrapper and are then handed to the verified solvers. Each problem is given a timeout of four hours and a memory limit of 4 GB. For infinite-horizon problems, we set a discount factor of $\lambda = 0.95$ and require an accuracy of $\epsilon = 0.05$ (for our verified policy iteration and Storm, $\epsilon = 0$). Finite-horizon MDPs are run with $\lambda = 1$ and a horizon of $N = 50$.

We compare our work to the probabilistic model checkers PRISM (Hinton et al. 2006) and Storm (Hensel et al. 2021), that we extended to support discount factors. We use PRISM's explicit representation mode and its floating-point arithmetic mode, while we use Storm in its exact mode. The point of this setup is to individually evaluate the performance impact of both precise arithmetic and verified data structures and algorithms. For part of the benchmarks, we use the values generated by a hand-written implementation of Gauss-Seidel value iteration in Rust as initial values for the verified algorithms. That way, the verified programs certify the unverified solutions to achieve better performance.

Results We give an overview of the results in Table 1. First, we observe that the verified implementations using floating-point arithmetic can often compete with PRISM. We assume that this is in part due to PRISM being a generalist optimized to handle factored systems. An exception here is policy iteration, because PRISM uses an approximate method to compute the value of the policy that is much faster than our exact method based on Gaussian elimination. Comparing our algorithms, we see that policy iteration is only viable for small problems, as the arithmetic complexity of Gaussian elimination grows cubically in the number of states. The Storm model checker provides the same exact results but solves more instances. Thus certification of the Storm results is a promising direction for future work. Using floating-point arithmetic, the optimized algorithms consistently outperform value iteration. When precise arithmetic is used, Gauss-Seidel value iteration becomes comparatively slow. This is because the fractions representing the value estimates become larger more quickly for in-place updates, as they grow already over the course of a single iteration.

Overall, the experiments show a large performance gap between floating-point and precise arithmetic, especially as the number of iterations increases. With each additional iteration, the precise representation of the current value estimates gets more complex. This prompted us to combine the unverified floating-point Rust implementation for infinite-horizon problems with the verified precise arithmetic: the resulting values from the unverified solver are provided as initial values to the verified precise arithmetic implementation. This way we get the best of both worlds: values that are formally guaranteed to be ϵ-optimal, with performance characteristics comparable to unverified implementations.

Conclusion and Discussion

Our work provides a comprehensive formalization of the basics of MDPs with rewards in the interactive theorem prover Isabelle/HOL. We then prove correct optimized algorithms to solve discounted MDPs. We show that it is feasible to solve non-trivial Markov decision processes with verified algorithms. Overall, our formalization is comprised of approximately 13.000 lines of definitions and proofs. One point worth highlighting is that we use the output of an unverified efficient implementation of value iteration as input to

[2] https://ipc2018-probabilistic.bitbucket.io/

	Instances	VI			GS			PI			MPI			Cert.			Fin-Horizon		
		Prism	Verified		Prism	Verified		Prism	Verified		Prism	Verified		VI	GS	Storm	Prism	Verified	
			$\mathbb{R}$	$\mathbb{F}$		$\mathbb{R}$	$\mathbb{F}$		$\mathbb{R}$	$\mathbb{F}$		$\mathbb{R}$	$\mathbb{F}$					$\mathbb{R}$	$\mathbb{F}$
academic-advising	2	–	–	1	–	–	1	–	–	–	–	1	1	1	1	–	1	1	1
crossing-traffic	4	4	4	4	4	2	4	4	2	2	4	4	4	4	4	4	4	4	4
elevators	8	5	2	6	5	1	6	5	2	2	5	2	6	6	6	5	7	2	6
game-of-life	3	3	–	3	3	–	3	3	–	3	3	–	3	3	–	–	3	–	3
manufacturer	2	2	–	2	2	–	2	2	–	1	2	1	2	2	2	1	2	1	2
push-your-luck	5	5	5	5	5	5	5	5	2	2	5	5	5	5	5	5	5	5	5
skill-teaching	8	6	4	7	6	3	7	6	2	4	6	4	8	6	6	4	6	4	6
triangle-tireworld	6	4	4	4	4	4	4	4	2	2	4	4	4	4	4	4	6	4	4
wildfire	1	–	–	1	–	–	1	–	–	–	–	–	1	1	1	–	–	–	1
wildlife-preserve	8	6	5	8	6	4	8	6	4	6	6	6	8	8	8	6	8	6	8

Table 1: A table showing the number of instances solved by different algorithms. The first column gives the number of instances per domain. Columns two to five show the performance of the PRISM implementations vs. our implementations of value, policy, Gauss-Seidel, and modified policy iteration, where $\mathbb{R}$ and $\mathbb{F}$ indicate precise and floating-point arithmetic respectively. The sixth column displays the results for Storm. The seventh column shows the performance of using an unverified implementation of value iteration followed by one last verified iteration. The last column shows the results for the finite-horizon case.

a verified implementation that uses precise arithmetic. This leads to a system with formal guarantees with similar performance characteristics as an unverified system. The results show the feasibility of verifying practical algorithms in the realm of reinforcement learning or probabilistic planning. In addition to an extensive library of existing formal proofs, Isabelle/HOL provides powerful facilities for proof search, automation, and structuring that in combination allow the development of verified software.

Future Work The most immediate extension of our work to handle much larger state spaces is to formalize data structures for factored representations of MDPs. These representations are standard in both model-checking and planning. Furthermore, in many applications, the current state of the environment is only partially known by the agent. Such systems are modeled using partially observable MDPs. Extending our work in this direction is an important step towards the verification of reinforcement learning algorithms. Another interesting direction concerns how to handle arithmetic: instead of relying on floating-point arithmetic which does not account for the accumulation of errors, we might investigate the possibility of using interval arithmetic to provide error-bounds. Lastly, verified Monte Carlo algorithms would allow us to deal with large state spaces.

Related Work There is a number of related verification approaches that have been tried in the general context of artificial intelligence. The first such thread of research is based on using completely automated methods. It has been applied in planning (Eriksson, Röger, and Helmert 2017) and SAT (Wetzler, Heule, and Hunt 2014), and has recently received the most attention in the area of verifying robustness of neural networks (Katz et al. 2017). In this latter application, a specification of a given neural network's robustness is compiled into an SMT formula, which is then automatically proved/disproved by an SMT solver. An advantage of this approach compared to using ITPs is that it is fully automated, i.e. one could prove a neural network safe without fully understanding it. A disadvantage, however, is that SMT solvers are limited in terms of what specifications they can automatically solve. This is most evident when trying to verify properties of algorithms or programs. For instance, they cannot prove a specification stating that a given implementation of value iteration computes an optimal policy.

Another related approach is that of (Selsam, Liang, and Dill 2017) and (Bagnall and Stewart 2019). In that thread of work, the authors develop ITP frameworks to aid in the formal reasoning about machine learning models and neural network architectures. Since they use ITPs, they are able to prove specifications that are more interesting than neural network robustness. For instance, they are able to show that a certain learning algorithm can guarantee a certain generalisation error, which cannot be done using SMT solvers.

MDPs have been formalized in theorem provers before, e.g. to analyze the semantics of pGCL (Hölzl 2017), or verify soundness of off-policy evaluation (Yeager et al. 2022). The most closely related formalization is the project by (Vajjha et al. 2021) who develop the library *CertRL* in the interactive theorem prover Coq. Our work differs in a number of ways: they show correctness of value and policy iteration while we also prove correct more involved algorithms. Furthermore, they do not verify efficient implementations from their formalisations, while we do. They also define the expected total discounted reward as equation (7), while we give a simpler and more natural definition, which needs the construction of the trace space of an MDP. We also show that deterministic decision rules are in fact optimal over all policies, while they only consider deterministic decision rules as candidates for optimal policies. Furthermore, in Isabelle/HOL there exists a recent formalization of the basics of discrete reinforcement learning (Chevallier and Fleuriot 2021). Their approach uses stochastic matrices instead of the Giry Monad. They also use equation (7) as their definition of the expected reward, only cover finite MDPs, do not discuss executable algorithms, and prove, at an abstract level, that value and policy iteration solve MDPs optimally.

Acknowledgements

This work was partially funded by the Deutsche Forschungsgemeinschaft Research Training Group CONVEY - 378803395/GRK2428 and the Deutsche Forschungsgemeinschaft Koselleck Grant NI 491/16-1. We thank Thomas Keller for helpful comments and providing us with benchmark problems.

References

Abdulaziz, M.; Gretton, C.; and Norrish, M. 2019. A Verified Compositional Algorithm for AI Planning. In *The 10th International Conference on Interactive Theorem Proving (ITP)*.

Abdulaziz, M.; and Lammich, P. 2018. A Formally Verified Validator for Classical Planning Problems and Solutions. In *The 30th International Conference on Tools with Artificial Intelligence (ICTAI)*.

Bagnall, A.; and Stewart, G. 2019. Certifying the True Error: Machine Learning in Coq with Verified Generalization Guarantees. In *The 33rd AAAI Conference on Artificial Intelligence (AAAI)*.

Ballarin, C. 2014. Locales: A Module System for Mathematical Theories. *J. Autom. Reason.*

Bellman, R. 1957. A Markovian Decision Process. *Journal of mathematics and mechanics*.

Chevallier, M.; and Fleuriot, J. D. 2021. Formalising the Foundations of Discrete Reinforcement Learning in Isabelle/HOL. *CoRR*.

Coen, C. S.; and Zoli, E. 2008. A Note on Formalising Undefined Terms in Real Analysis. In *International Workshop on Proof Assistants and Types in Education (PATE)*.

Eriksson, S.; Röger, G.; and Helmert, M. 2017. Unsolvability Certificates for Classical Planning. In *The 27th International Conference on Automated Planning and Scheduling (ICAPS)*.

Esparza, J.; Lammich, P.; Neumann, R.; Nipkow, T.; Schimpf, A.; and Smaus, J.-G. 2013. A Fully Verified Executable LTL Model Checker. In *25th International Conference on Computer Aided Verification (CAV)*.

Giry, M. 1982. A Categorical Approach to Probability Theory. In *Categorical Aspects of Topology and Analysis*.

Haftmann, F.; and Nipkow, T. 2007. A Code Generator Framework for Isabelle/HOL. Technical report, Department of Computer Science, University of Kaiserslautern.

Haftmann, F.; and Wenzel, M. 2006. Constructive Type Classes in Isabelle. In *The International Workshop on Types for Proofs and Programs (TYPES)*.

Hensel, C.; Junges, S.; Katoen, J.-P.; Quatmann, T.; and Volk, M. 2021. The probabilistic model checker Storm. *International Journal on Software Tools for Technology Transfer*, 1–22.

Hinton, A.; Kwiatkowska, M. Z.; Norman, G.; and Parker, D. 2006. PRISM: A Tool for Automatic Verification of Probabilistic Systems. In *The 22nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*.

Hölzl, J. 2013. *Construction and Stochastic Applications of Measure Spaces in Higher-Order Logic*. Ph.D. thesis, Technical University Munich.

Hölzl, J. 2017. Markov Chains and Markov Decision Processes in Isabelle/HOL. *J. Autom. Reason.*

Katz, G.; Barrett, C. W.; Dill, D. L.; Julian, K.; and Kochenderfer, M. J. 2017. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In *29th International Conference Computer Aided Verification (CAV)*.

Klein, G.; Elphinstone, K.; Heiser, G.; Andronick, J.; Cock, D.; Derrin, P.; Elkaduwe, D.; Engelhardt, K.; Kolanski, R.; Norrish, M.; Sewell, T.; Tuch, H.; and Winwood, S. 2009. seL4: Formal Verification of an OS Kernel. In *22nd ACM Symposium on Operating Systems Principles 2009 (SOSP)*.

Kuncar, O.; and Popescu, A. 2016. From Types to Sets by Local Type Definitions in Higher-Order Logic. In *The 7th International Conference on Interactive Theorem Proving (ITP)*.

Leroy, X. 2009. Formal Verification of a Realistic Compiler. *Commun. ACM*.

Milner, R. 1972. Logic for Computable Functions Description of a Machine Implementation. Technical report, Stanford University.

Nipkow, T.; Eberl, M.; and Haslbeck, M. P. L. 2020. Verified Textbook Algorithms - A Biased Survey. In *The 18th International Symposium on Automated Technology for Verification and Analysis (ATVA)*.

Nipkow, T.; Paulson, L. C.; and Wenzel, M. 2002. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer.

Paulson, L. C.; and Blanchette, J. C. 2010. Three Years of Experience with Sledgehammer, a Practical Link between Automatic and Interactive Theorem Provers. In *PAAR@ IJCAR*.

Puterman, M. L. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Series in Probability and Statistics. Wiley.

Selsam, D.; Liang, P.; and Dill, D. L. 2017. Developing Bug-Free Machine Learning Systems With Formal Mathematics. In *The 34th International Conference on Machine Learning (ICML)*.

Thiemann, R.; and Yamada, A. 2016. Formalizing Jordan Normal Forms in Isabelle/HOL. In *The 5th ACM SIGPLAN Conference on Certified Programs and Proofs (CPP)*.

Vajjha, K.; Shinnar, A.; Trager, B. M.; Pestun, V.; and Fulton, N. 2021. CertRL: Formalizing Convergence Proofs for Value and Policy Iteration in Coq. In *The 10th International Conference on Certified Programs and Proofs (CPP)*.

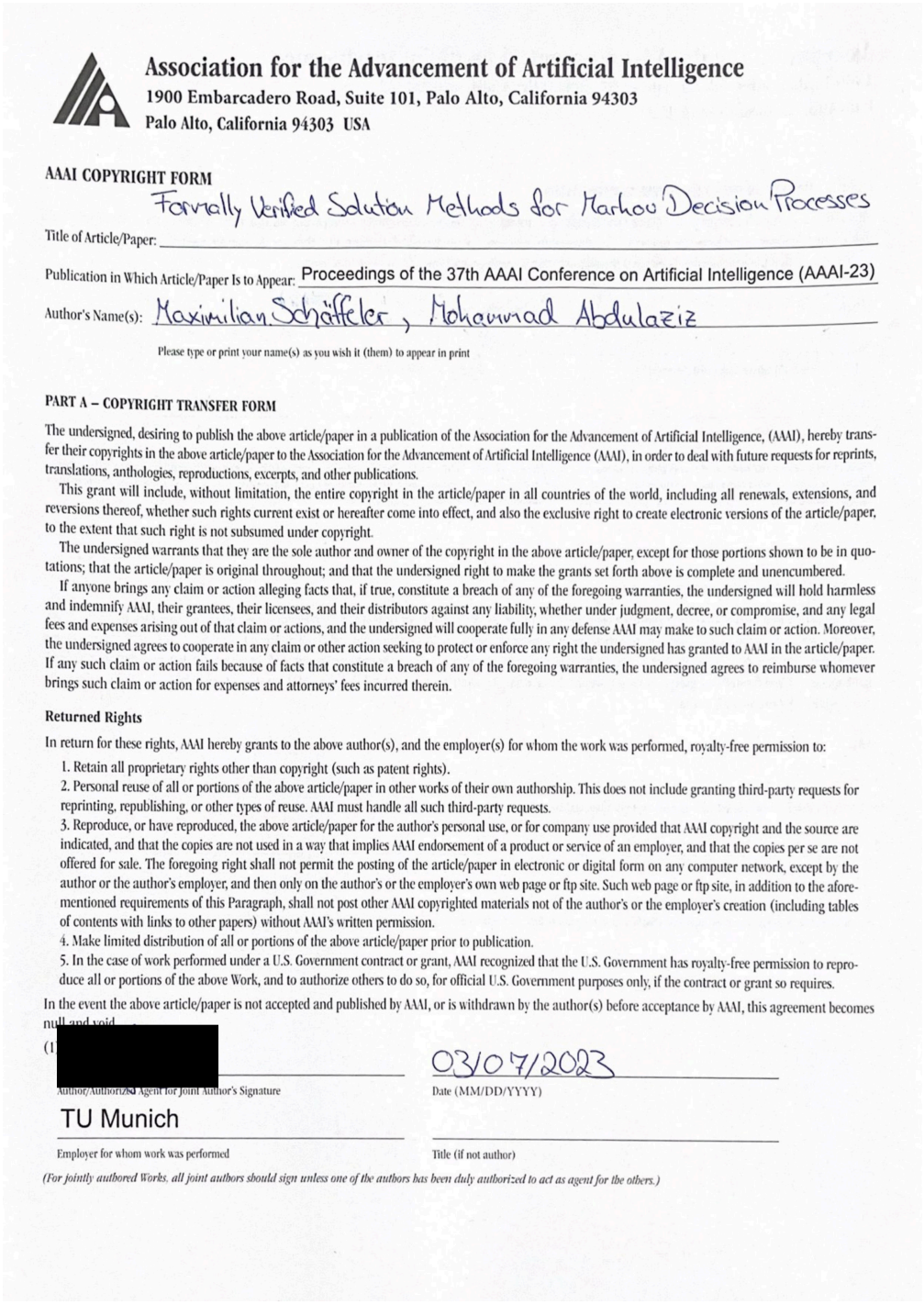
Wenzel, M. 1999. Isar - A Generic Interpretative Approach to Readable Formal Proof Documents. In *Theorem Proving in Higher Order Logics, 12th International Conference, TPHOLs'99, Proceedings*. Springer.

Wetzler, N.; Heule, M.; and Hunt, W. A., Jr. 2014. DRAT-trim: Efficient Checking and Trimming Using Expressive Clausal Proofs. In *International Conference on Theory and Applications of Satisfiability Testing (SAT)*.

Wirth, N. 1971. Program Development by Stepwise Refinement. *Commun. ACM*.

Yeager, J.; Moss, J. E. B.; Norrish, M.; and Thomas, P. S. 2022. Mechanizing Soundness of Off-Policy Evaluation. In *13th International Conference on Interactive Theorem Proving, ITP 2022*.

Copyright and License Information



Association for the Advancement of Artificial Intelligence
1900 Embarcadero Road, Suite 101, Palo Alto, California 94303
Palo Alto, California 94303 USA

AAAI COPYRIGHT FORM

Title of Article/Paper: Formally Verified Solution Methods for Markov Decision Processes

Publication in Which Article/Paper Is to Appear: Proceedings of the 37th AAAI Conference on Artificial Intelligence (AAAI-23)

Author's Name(s): Maximilian Schäffeler, Mohammad Abdulaziz

Please type or print your name(s) as you wish it (them) to appear in print

PART A – COPYRIGHT TRANSFER FORM

The undersigned, desiring to publish the above article/paper in a publication of the Association for the Advancement of Artificial Intelligence, (AAAI), hereby transfer their copyrights in the above article/paper to the Association for the Advancement of Artificial Intelligence (AAAI), in order to deal with future requests for reprints, translations, anthologies, reproductions, excerpts, and other publications.

This grant will include, without limitation, the entire copyright in the article/paper in all countries of the world, including all renewals, extensions, and reversions thereof, whether such rights current exist or hereafter come into effect, and also the exclusive right to create electronic versions of the article/paper, to the extent that such right is not subsumed under copyright.

The undersigned warrants that they are the sole author and owner of the copyright in the above article/paper, except for those portions shown to be in quotations; that the article/paper is original throughout; and that the undersigned right to make the grants set forth above is complete and unencumbered.

If anyone brings any claim or action alleging facts that, if true, constitute a breach of any of the foregoing warranties, the undersigned will hold harmless and indemnify AAAI, their grantees, their licensees, and their distributors against any liability, whether under judgment, decree, or compromise, and any legal fees and expenses arising out of that claim or actions, and the undersigned will cooperate fully in any defense AAAI may make to such claim or action. Moreover, the undersigned agrees to cooperate in any claim or other action seeking to protect or enforce any right the undersigned has granted to AAAI in the article/paper. If any such claim or action fails because of facts that constitute a breach of any of the foregoing warranties, the undersigned agrees to reimburse whomever brings such claim or action for expenses and attorneys' fees incurred therein.

Returned Rights

In return for these rights, AAAI hereby grants to the above author(s), and the employer(s) for whom the work was performed, royalty-free permission to:

1. Retain all proprietary rights other than copyright (such as patent rights).
2. Personal reuse of all or portions of the above article/paper in other works of their own authorship. This does not include granting third-party requests for reprinting, republishing, or other types of reuse. AAAI must handle all such third-party requests.
3. Reproduce, or have reproduced, the above article/paper for the author's personal use, or for company use provided that AAAI copyright and the source are indicated, and that the copies are not used in a way that implies AAAI endorsement of a product or service of an employer, and that the copies per se are not offered for sale. The foregoing right shall not permit the posting of the article/paper in electronic or digital form on any computer network, except by the author or the author's employer, and then only on the author's or the employer's own web page or ftp site. Such web page or ftp site, in addition to the aforementioned requirements of this Paragraph, shall not post other AAAI copyrighted materials not of the author's or the employer's creation (including tables of contents with links to other papers) without AAAI's written permission.
4. Make limited distribution of all or portions of the above article/paper prior to publication.
5. In the case of work performed under a U.S. Government contract or grant, AAAI recognized that the U.S. Government has royalty-free permission to reproduce all or portions of the above Work, and to authorize others to do so, for official U.S. Government purposes only, if the contract or grant so requires.

In the event the above article/paper is not accepted and published by AAAI, or is withdrawn by the author(s) before acceptance by AAAI, this agreement becomes null and void.

(1 [redacted]

Author/Authorized Agent for Joint Author's Signature

03/07/2023

Date (MM/DD/YYYY)

TU Munich

Employer for whom work was performed

Title (if not author)

(For jointly authored Works, all joint authors should sign unless one of the authors has been duly authorized to act as agent for the others.)

AAAI copyright form

Re: Permission to Include AAAI Paper in Cumulative Thesis

Subject: Re: Permission to Include AAAI Paper in Cumulative Thesis
From: AAAI Proceedings Questions <proceedings-questions@aaai.org>
Date: 17.01.25, 07:01
To: Maximilian Schaeffeler <maximilian.schaeffeler@tum.de>

Dear Maiximilian,
You don't need any special permission to use your PDF in your dissertation. AAAI returns the right to you on this case.
Best regards,
Francisco Cruz,
AAAI Press Editorial Team

On Thu, Jan 16, 2025 at 3:35 PM Maximilian Schaeffeler <maximilian.schaeffeler@tum.de> wrote:

> Good Morning,
>
> I am seeking clarification regarding the use of articles authored by myself in a cumulative dissertation/doctoral thesis.
> Is this right granted as part of the AAAI copyright form or am I required to file a "Request to Reproduce Copyrighted Materials"?
> Specifically, I would need to include the full PDF of my 2023 AAAI paper "Formally Verified Solution Methods for Markov Decision Processes".
>
> Thank you for your assistance. I look forward to your clarification.
>
> Best regards,
>
> Maximilian Schäffeler
>
>

AAAI copyright clarification

Core Publication 2

Maximilian Schäffeler and Mohammad Abdulaziz. "Formally Verified Approximate Policy Iteration." In: *AAAI Conference on Artificial Intelligence*. Vol. 39. AAAI Press, 2025. To appear.

Formally Verified Approximate Policy Iteration

Maximilian Schäffeler[1], Mohammad Abdulaziz[2]

[1]Technische Universität München, Germany
[2]King's College London, United Kingdom
maximilian.schaeffeler@tum.de, mohammad.abdulaziz@kcl.ac.uk

Abstract

We present a methodology based on interactive theorem proving that facilitates the development of verified implementations of algorithms for solving factored Markov Decision Processes. As a case study, we formally verify an algorithm for approximate policy iteration in the proof assistant Isabelle/HOL. We show how the verified algorithm can be refined to an executable, verified implementation. Our evaluation on benchmark problems shows that it is practical. As part of the development, we build verified software to certify linear programming solutions. We discuss the verification process and the modifications we made to the algorithm during formalization.

Code — https://github.com/schaeffm/fmdp_isabelle

Introduction

Markov Decision Processes (MDPs) are models of probabilistic systems, with applications in AI, model checking, and operations research. In AI, for instance, given a description of the world in terms of states and actions that can change those states in a randomised fashion, one seeks a *policy* that determines the actions chosen in every state, with the aim of accruing maximum *reward*. There is a large number of methods to solve MDPs, most notably, value and policy iteration, which compute policies with optimality guarantees.

In many applications in AI or autonomous systems (Lahijanian et al. 2010; Junges et al. 2018), obtaining an optimal policy is safety-critical, with the goal of e.g. minimizing the number of accidents. One important aspect here is the assurance that the output of the MDP solving system is correct. Such assurance is currently attained to some degree by testing and other software engineering methods. However, the best guarantee can be achieved by mathematically proving the MDP solver and the underlying algorithm correct. A successful way of mathematically proving correctness properties of (i.e. formally verifying) pieces of software is using Interactive Theorem Provers (ITPs), which are formal mathematical systems that one can use to devise machine-checked proofs. Indeed, ITPs have been used to prove correctness properties of compilers (Leroy 2009), operating systems kernels (Klein et al. 2009), model checkers (Esparza et al. 2013), planning systems (Abdulaziz and Lammich 2018; Abdulaziz and Koller 2022; Abdulaziz and Kurz 2023), and, most related to the topic of this work, algorithms to solve MDPs (Schäffeler and Abdulaziz 2023). A challenge with using ITPs to prove algorithms correct, nonetheless, is that they require intense human intervention. Thus for an ITP to be successfully employed in a serious verification effort, novel ideas in the design of the software to be verified as well as the underlying proof have to be made.

In this paper, we consider formally verifying algorithms for solving *factored* MDPs. A challenge to using MDPs to model realistic systems is their, in many cases, enormous size. For such systems, MDPs are succinctly represented as factored MDPs. The system's state is characterised as an assignment to a set of state variables and actions are represented in a compact way by exploiting the structure present in the system. Such representations are common in AI (Guestrin et al. 2003; Sanner 2010; Younes and Littman 2004) and in model checking (Hinton et al. 2006; Dehnert et al. 2017). Although ITPs have been used prove the correctness of multiple types of software and algorithms, including algorithms on MDPs, algorithms on factored MDPs are particularly challenging. The root of this difficulty is that the succinctness of the representation comes at a cost. Naively finding a solution for a factored MDP could entail the construction of structures exponentially bigger than the factored MDP. This necessitates using advanced data structures, heuristics and computational techniques, to avoid that full exponential blow up.

Our main contribution is that we develop a methodology based on using the Isabelle *locale* system, to structure the MDP solving algorithm into parts amenable to verification. To enable this methodology, we build a formal mathematical library allowing the specification of algorithms for solving factored MDPs and their properties, like algorithms for planning under uncertainty and probabilistic model checking. We also develop a number of reusable building blocks to be used in other algorithms, e.g. a certificate checker for linear programming solutions. Potential targets for formalization include probabilistic model checking, planning and reinforcement learning algorithms (Hartmanns et al. 2023; Keller and Eyerich 2012). We describe the methodology in terms of the verification of Guestrin et al.'s approximate pol-

icy iteration algorithm in the Isabelle theorem prover.

This algorithm computes approximate policies, i.e. suboptimal policies with guarantees on their optimality, for one type of factored MDPs. The algorithm we consider combines scoped functions and decision lists, which are data structures that exploit the factored representation, linear programming, in addition to probabilistic reasoning and dynamic programming. The combination of this wide range of mathematical/algorithmic concepts and techniques is what makes this algorithm particularly hard from a verification perspective. To get an idea of the scale, we advise the reader to look at Fig. 2, which shows the hierarchy of concepts and definitions which we had to develop (aka *formalize*) within Isabelle/HOL to be able to state the algorithm and prove its correctness statement. This is, of course, in addition to the notions of analysis, probabilities, and MDPs, which already exist in Isabelle/HOL. Furthermore, to be able to prove the algorithm correct, we had to design an architecture of the implementation that makes verification feasible. Our architecture mixes verification and certification: we verify the entire algorithm, and build a verified certificate checker for linear programming solutions that delivers formal guarantees. In addition to proving the algorithm correct, we obtain a formally verified implementation, that we experimentally show to be practical. Our work, as far as we are aware, is the first work on formally verifying algorithms for factored MDPs.

Background

We introduce the interactive theorem prover Isabelle/HOL, our formal development of factored MDPs relate it to existing formalizations in Isabelle/HOL.

Isabelle/HOL

An ITP is a program that implements a formal mathematical system, in which definitions and theorem statements can be expressed, and proofs are constructed from a set of axioms. To prove a fact in an ITP, the user only provides the high-level steps while the ITP fills in the details at the level of axioms. Specifically, our developments use the interactive theorem prover Isabelle/HOL (Nipkow, Paulson, and Wenzel 2002) based on Higher-Order Logic, a combination of functional programming with logic. Isabelle is highly trustworthy, as the basic inference rules are implemented in a small, isolated kernel. Outside the kernel, several tools implement proof tactics, data types, recursive functions, etc..

Our presentation of definitions and theorems here deviates slightly from the formalization in Isabelle/HOL. Specifically, we use subscript notation for list indexing and some function applications. We use parentheses for function application. For a list xs, $len(xs)$ returns the length, while $xs_{:i}$ returns the first i elements of xs. List concatenation is written as $xs{\cdot}ys$, $x :: xs$ inserts x at the front of the list xs, $map(f, xs)$ applies f to every element of xs. Finally, $\mathbb{N}_{<i}$ is short for the first i natural numbers (including 0).

Factored MDPs

Factored MDPs are compactly represented MDPs that exploit regularities in large MDPs, which can often lead to an exponential reduction in the size of the model. Common formats to store factored MDPs include JANI (Budde et al. 2017), the PRISM language (Hinton et al. 2006), and RDDL (Sanner 2010). We implement the factoring described in (Guestrin et al. 2003). In Isabelle/HOL, we define factored MDPs using *locales* (Ballarin 2014). A locale introduces a mathematical context with constants and assumptions, in which we develop our formalization. Locales can be instantiated with concrete constants and a proof that discharges the assumptions of the locale, yielding all the theorems proved within the locale. For example, we instantiate the MDP locale from Schäffeler and Abdulaziz with the factored systems introduced in this section, and are therefore able to reuse important definitions and theorems. Moreover, reducing factored MDPs to a tested and reviewed library enhances confidence in our definitions.

State Space A state of a factored MDP is an assignment of values to its n state variables. Each state variable $i \in \mathbb{N}_{<n}$ has a finite, nonempty domain X_i. In Isabelle/HOL, we implement an MDP state x as a map, i.e. a function with an explicit domain $dom(x)$. A *partial state* is a map where only a subset of the state variables are assigned, but all entries are valid, i.e. $dom(x) \subseteq \mathbb{N}_{<n}$ and $x_i \in X_i$ for $i \in dom(x)$. The set X of *states* of the MDP consists of all partial states x where $dom(x) = \mathbb{N}_{<n}$. The domain of a state x can be restricted to a set of variables Y, denoted as $x|_Y$. One partial state x is called *consistent* with another partial state t (written $x \sqsubseteq t$), if and only if $x|_{dom(t)} = t$.

Example As a running example, we use a model of a computer network with ring topology from Guestrin et al.'s original paper (see Fig. 1). In the ring, each machine is either working or broken, and states change stochastically. Each machine C_i's state of operation is characterized by a variable i, s.t. all domains $X_i = \{\mathrm{W}, \mathrm{B}\}$. In a ring with three machines, a partial state is $s := [1 \mapsto \mathrm{W}, 3 \mapsto \mathrm{B}]$, $dom(s) = \{1, 3\}$. It holds that $s \sqsubseteq [1 \mapsto \mathrm{W}]$, but $s \not\sqsubseteq [2 \mapsto \mathrm{B}]$.

Scoped Functions For many MDPs, the transition behavior and the rewards can be computed from a combination of functions that individually only depend on a small subset of all state variables. Such *scoped functions* take a partial state as input and determine the output inspecting only variables within their scope. The algorithm we formalize expresses policy iteration using scoped functions, which avoids enumerating the full state space. Scoped functions pose a challenge for formalization, as scopes could be represented implicitly or explicitly: we can either prove that a given function has a restricted scope, or the function can store its scope explicitly. In Isabelle/HOL we do both: decoupling scopes from the function definition is more flexible, as one may derive multiple scopes for a function. However, since it is in general infeasible to compute the precise scope of a function, we use explicit scopes in the executable version. An important operation on scoped functions is the instantiation with a partial state t: $inst_t$ applied to a scoped function returns a new scoped function with reduced scope, where all input dimensions that t provides are fixed to the values of t.

(a) (b)

$X \mid_{scope(\mathcal{P}_1^d)}$	$\mathcal{P}_1^d(X, X'_1 \mapsto \text{W})$
$X_1 \mapsto$ W, $X_3 \mapsto$ W	0.9
$X_1 \mapsto$ B, $X_3 \mapsto$ W	0.2
$X_1 \mapsto$ W, $X_3 \mapsto$ B	0.7
$X_1 \mapsto$ B, $X_3 \mapsto$ B	0.1

(c)

Figure 1: (a) Ring network of 3 machines. (b) Variable dependencies of the default action. (c) Probabilities of C_1 working in the next step, for every state of C_1 and C_3.

Transitions In our setting, a factored MDP comes with a finite set of actions A, and a default action $d \in A$. For each action $a \in A$, transition probabilities $\mathcal{P}_i^a : X \to \mathbb{P}(X_i)$ with $scope(\mathcal{P}_i^a) \subseteq \mathbb{N}_{<n}$ determine the evolution of variable i. Here, $\mathbb{P}(X)$ denotes the set of probability distributions over a finite set X. The set $effects_a$ defines the state variables where the behavior of a differs from the default action (i.e. variables i s.t. $\mathcal{P}_i^a \neq \mathcal{P}_i^d$). The combined transition probabilities between two states x and x' are defined as $\mathcal{P}^a(x, x') \coloneqq \prod_{i<n} \mathcal{P}_i^a(x, x'_i)$.

Example In the ring topology domain, for each machine there is an action to restart it, in which case it is guaranteed to work in the next step. On the other hand, the default action d is to do nothing. In general, the probability of a machine working in the next step depends on its own state and the state of the predecessor, e.g. $scope(\mathcal{P}_2^d) = \{1, 2\}$. The exact conditional probability distribution for the MDP's evolution under the default action is shown in Fig. 1. Using an explicit representation to model this factored action, we would need 8 transitions, each consisting of a distribution over 8 possible successor states. This is in contrast to three tables in the factored case. A common way to model the transition behavior is using dependency graphs as shown in Fig. 1.

Rewards The actions $a \in A$ define scoped reward functions $R_i^a : X \to \mathbb{R}$ for $i < r_a$. The reward for selecting a is a sum of those reward functions: $R^a(x) \coloneqq \sum_{i<r_a} R_i^a(x)$. We assume that the first r_d reward functions are the same for all actions. Now, given a policy $\pi : X \to A$ we are interested in the discounted expected total reward $\nu_\pi(x) \coloneqq \mathbb{E}_{\omega \sim \mathcal{T}(\pi,x)}\left[\sum_i \gamma^i R^{\pi(\omega_i)}(\omega_i)\right]$ with discount factor $\gamma < 1$ and trace space $\mathcal{T}$. Our goal is to achieve the optimal reward $\nu^*(x) \coloneqq \sup_{\pi \in \Pi} \nu_\pi(x)$. For a value estimate $v : X \to \mathbb{R}$ and an action a, the one-step lookahead is defined as

$$Q_v^a(x) \coloneqq R^a(x) + \gamma \sum_{x' \in X} \mathcal{P}^a(x, x') \cdot v(x').$$

For each state x, the maximum lookahead w.r.t. all actions is $Q_v^*(x)$. A policy π is called *greedy* if Q_v^π and Q_v^* are equal for all states. The Bellman error, denoted by $\|v - Q_v^\pi\|$, is the maximum difference between Q_v^π and v, over all states in X, i.e. the L_∞ distance.

Algorithm 1: Approximate Policy Iteration (API)

$api(t, \pi, w) \coloneqq$
  if $t \geq t_{max}$ **or** $err \leq \epsilon$ **or** $w_=$
    then $(t, \pi', w', err, w_=)$
    else $api(t + 1, \pi', w')$
  where $w' \coloneqq upd\_w(\pi)$
    $\pi' \coloneqq greedy\_\pi(w')$
    $err \coloneqq factored\_err(\pi', w')$
    $w_= \coloneqq w' = w$

Example In our example, a factored representation of the rewards R_i^d is $R_i^d(\text{W}) = 1$ and $R_i^d(\text{B}) = 0$, for all $1 \leq i \leq 3$. This gives rise to an exponentially smaller representation compared to an explicitly represented MDP, where the reward function for d would have 8 entries.

Linear Value Functions

Even if all reward and transition functions are scoped functions, the value function ν_π may still be unstructured, i.e. computing ν_π might require the construction of an exponentially big mapping (Guestrin et al. 2003). However, the value of a policy can be approximated as the weighted sum of m basis functions $h_i : X \to \mathbb{R}$. Given weights w_i for each h_i, the value of a state x is defined as a weighted sum $\nu_w(x) \coloneqq \sum_{i<m} w_i h_i(x)$. Note that the efficiency of the algorithm we verify here crucially depends on the fact that h_i is a function with small scope.

For each action choice a and basis function h_i, we can compute its expected evaluation g_i^a, defined as $\sum x' \in X.\ \mathcal{P}^a(x, x') \cdot h_i(x')$, in the successor state. As g is independent of the concrete weights, it can be computed once for a set of basis functions, and is then cached for efficiency. In Isabelle/HOL, we prove that $g_i^a(x)$ has a structured representation with scope $\Gamma_i^a \coloneqq \bigcup_{j \in scope(h_i)} .\ scope(\mathcal{P}_j^a)$. This also leads to an efficient computation of the Q functions:

$$Q_w^a(x) \coloneqq Q_{\nu_w}^a(x) = R^a(x) + \gamma \sum_{i<m} w_i g_i^a(x).$$

Example The value functions in the ring domain can be approximately represented using the basis function $h_0 = 1$ and one function per machine: $h_i = 1$ if $X_i = \text{W}$ and 0 otherwise, for $1 \leq i \leq 3$. Note that since these basis functions have very limited scopes, the accuracy of the best possible approximation of the value function is also limited.

Approximate Policy Iteration

Approximate Policy Iteration (API) is a variant of policy iteration that exploits structure in MDPs to scale to large systems (Guestrin et al. 2003). API is an iteration algorithm, where each iteration consists of three parts: policy evaluation, policy improvement and Bellman error computation. The algorithm terminates when either a timeout t_{max} is reached, the error dips below a threshold ϵ, or the weights assigned to the basis functions converge. In Isabelle/HOL, the algorithm is implemented as a function $api(t, \pi, w)$ (Alg. 1),

Algorithm 2: Decision List Policy

$$
\begin{aligned}
&\mathit{greedy\_}\pi \coloneqq \\
&\quad \mathit{sort\_}\pi((\bot_M, d, 0) :: \mathit{concat}([\pi_a \mid a \in A - \{d\}]) \\
&\quad \textbf{where } \pi_a \coloneqq [(x, a, \delta_a(x)) \mid \delta_a(x) > 0, x \in X_{\mid \mathbf{T}_a}] \\
&\qquad\quad \delta_a \coloneqq Q^a_w - Q^d_w \\
&\qquad\quad \mathbf{T}_a \coloneqq \mathit{scope}(R^a) \cup \textstyle\bigcup_{i \in \mathbf{I}_a} \Gamma^a_i \cup \Gamma^d_i \\
&\qquad\quad \mathbf{I}_a \coloneqq \{i < m \mid \mathit{effects}_a \cap \mathit{scope}(h_i) \neq \emptyset\}
\end{aligned}
$$

that takes as inputs a time step t, weights for the basis functions w and a policy π. The initial call to the algorithm is $api(0, \pi^0, w^0)$ where $w_0 = 0$ and π^0 is some greedy policy w.r.t. w^0. An iteration of API first uses the current policy π to first compute updated weights w', then a new greedy policy π', and finally the Bellman error err of π'. If the termination condition is met, the algorithm returns the current iteration, weights and policy, as well as the error and whether the weights converged. Otherwise, api is called recursively.

We structure and decouple the algorithm using locales. Conceptually, this usage of locales is similar to using modules in programming languages. Here, we use locales to postulate the existence of three functions ($\mathit{upd\_w}$, $\mathit{greedy\_}\pi$, $\mathit{factored\_err}$) along with their specifications:

Specification 1 ($\mathit{upd\_w}$)**.** A decision list policy a list representation of policies where each entry (called branch) is a pair (t, a) of a partial state and an action. To select an action in a state x, we search the list for the first branch where x is consistent with t. Now fix a decision list policy π, let $w' = \mathit{upd\_w}(\pi)$. Then $\nu_{w'}$ is the best possible estimate of ν_π: $\|\nu_{w'} - \nu_\pi\| = \inf_w \|\nu_w - \nu_\pi\|$.

Specification 2 ($\mathit{greedy\_}\pi$)**.** For all weights w, $\mathit{greedy\_}\pi(w)$ is a greedy decision list policy for ν_w.

Specification 3 ($\mathit{factored\_err}$)**.** Given weights w and a greedy decision list policy π for ν_w, $\mathit{factored\_err}$ determines the Bellman error: $\mathit{factored\_err}(\pi, w) = \|Q^*_w - \nu_w\|$.

For now, we merely state specifications for the algorithms API builds upon, only later will we show how to implement the specifications concretely. This approach keeps the assumptions on individual parts of the algorithm explicit and permits an easier exchange of implementations, e.g. in our developments one may swap the LP certification algorithm for a verified LP solver implementation. It also facilitates gradual verification of software: the correct behavior of the software system can be proved top-down starting from assumptions on each component. In the following sections we show that these specifications have efficient implementations. See Fig. 2 for an overview of all components.

The choice of the specifications and the decomposition of the algorithm into components is roughly based on the presentation of the algorithm by Guestrin et al.. We identified the above specifications after multiple iterations. There is usually a trade-off between the simplicity of the specification and the complexity of the implementation: the smaller the internal complexity, the more complex the external complexity, i.e. the interactions between algorithms.

Within the context of the locale, assuming all specifications, we can derive the same error bounds as presented by Guestrin et al.. One exemplary important observation is that if the weights converge during API, then in the last step the Bellman error equals the approximation error. This leads to the following *a posteriori* optimality bound:

Algorithm 3: Factored Bellman Error

$$
\begin{aligned}
&\mathit{factored\_err} \coloneqq \\
&\quad \sup_{i < \mathit{len}(\pi)} \mathit{branch\_err}(\pi_i, \mathit{map}(\mathit{fst}, \pi_{:i}))
\end{aligned}
$$

Theorem 1. *Let* $api(w_0, \pi_0) = (t', \pi, w, err, \mathit{True})$. *Then* $(1 - \gamma)\|\nu^* - \nu_w\| \leq 2\gamma \cdot err$.

Policy Improvement

Given weights w, the policy improvement phase determines a greedy policy w.r.t. ν_w. The policy takes the form of a decision list, where each element is a pair of a partial state and an action. The main idea for an efficient computation is to only consider actions better than the default action. This notion is made precise by the bonus function δ_a (Alg. 2) with scope $\mathbf{T}_a$. Guestrin et al. do not include Γ^d in $\mathbf{T}_a$, which we assume to be an oversight in the definition: the behavior of the default action does influence the bonus. Unless components cancel out, the scope of a function difference is the union of the scopes of both arguments. Finally, we concatenate the branches π_a for every action but d, add the default action as a fallback and sort the decision list policy by decreasing bonus. Here, the empty map with no entries is $\bot_M$. We can show that $\mathit{greedy\_}\pi$ satisfies Spec. 2, as action selection proceeds in the order of decreasing bonus.

Factored Bellman Error

The Bellman error $\|Q^*_w - \nu_w\|$ is an indicator of the degree of optimality of a policy. An inefficient computation would enumerate every state, and return the maximum error. However, for a decision list policy, we can compute the error incurred by each branch separately. The total error then equals the maximum error of any branch (Alg. 3). For now, assume that we have a function $\mathit{branch\_err}$ that computes the error for a single branch, i.e. the maximum error for any state that selects the respective branch. These are all states that are consistent with the current branch t, but did not match any prior branch $t' \in ts$ of the policy, formally $X_{(t,ts)} \coloneqq \{x \in X.\ t \sqsubseteq x \wedge \forall t' \in ts.\ t' \not\sqsubseteq x\}$. Hence, we also need to pass the prefix of the decision list policy to $\mathit{branch\_err}$. Note that if the branch is selected by no state its error defined as $-\infty$. We show that if Spec. 4 is met and π is a greedy policy w.r.t. w, then $\mathit{factored\_err}$ satisfies Spec. 3.

Specification 4 ($\mathit{branch\_err}$)**.** Given a prefix of a policy π, i.e. the current branch (t, a), and a list of partial states ts from prior branches, $\mathit{branch\_err}(t, a, ts) = \sup_{x \in X_{(t,ts)}} |Q^a_w(x) - \nu_w(x)|$.

Branch Error Consider the branch (t, a) of the policy, for a partial state t an action a. The states of all prior branches form the list of partial states ts. To find the Bellman error of

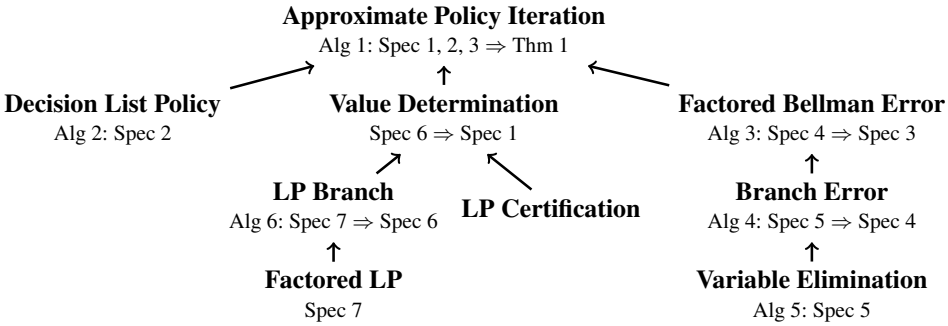


Figure 2: Simplified Locale Hierarchy

Algorithm 4: Branch Error

$$\begin{aligned} &branch\_err \coloneqq \max(max_\Sigma(fs \cdot \mathcal{I}'), max_\Sigma(-fs \cdot \mathcal{I}')) \\ &\quad \textbf{where } rs \coloneqq [R^a_0, \ldots, R^a_{r_a}] \\ &\qquad\quad ws \coloneqq [w_0(h_0 - \gamma g_0), \ldots, w_m(h_m - \gamma g_m)] \\ &\qquad\quad fs \coloneqq map(inst_t, rs \cdot ws) \end{aligned}$$

Algorithm 5: Variable Elimination

$$\begin{aligned} &max\_step(i, fs) \coloneqq (i+1, e :: E') \\ &\quad \textbf{where} \quad e \coloneqq x \mapsto \max_{y \in X_{\mathcal{O}(i)}} \sum_{f \in E} f(x_{\mathcal{O}(i) \mapsto y}) \\ &\qquad (E, E') \coloneqq partition(f \mapsto \mathcal{O}(i) \in scope(f), fs) \\ &max_\Sigma(fs) \coloneqq \textstyle\sum_{f \in fs'} f(\bot_M) \\ &\quad \textbf{where } (\_, fs') \coloneqq max\_step^n(0, fs) \end{aligned}$$

the current branch, we need to maximize $|Q^a_w(x) - \nu_w(x)|$ w.r.t. states $x \in X_{(t,ts)}$. Note that for all states x

$$Q^a_w(x) - \nu_w(x) = \sum_{i<r_a} R^a_i(x) + \sum_{i<m} w_i(h_i - \gamma g_i)(x). \quad (1)$$

Suppose we had an algorithm to efficiently compute the maximum sum of scoped functions. In that case we could determine the error of a branch. Again, we specify an algorithm max_Σ doing exactly that (Spec. 5). In Alg. 4, we call max_Σ with the functions from (1). To restrict the maximization to the states $X_{(t,ts)}$, we instantiate all functions with the partial state t. Additionally, we define the functions $\mathcal{I}'$ that evaluate to $-\infty$ on states that select a different branch of the policy. Hence, these states are ignored in the error computation. We also apply max_Σ to the negated functions to compute the absolute value of the error. Finally, we formally prove that $branch\_err$ satisfies Spec. 4.

Specification 5 (Variable Elimination)**.** For scoped functions fs, $max_\Sigma(fs) = \sup_{x \in X} \sum_{f \in fs} f(x)$.

Variable Elimination The specification for max_Σ can be efficiently implemented with a variable elimination algorithm (Alg. 5). In each iteration, the algorithm selects a dimension of the state space, collects all functions that depend on this dimension in a set E. It then creates a new function e that maximizes all functions in E over that dimension of the state space. The algorithm keeps track of a set of functions to maximize and the number of the current iteration. Since the number of operations performed by the algorithm varies greatly with the elimination order, the variables can be reordered with a bijection $\mathcal{O} : \mathbb{N}_{<n} \to \mathbb{N}_{<n}$. We formally prove that max_step preserves the maximum of the current list of functions, thus max_Σ meets Spec. 5.

Value Determination

After a new candidate policy is found, it is evaluated in the value determination phase. Since the exact value function can not in general be represented as a linear combination of the basis functions, we aim to find weights for the basis functions that minimize the approximation error (see Spec. 1), for which we need to solve a linear program (LP). The structure of the algorithm that finds optimal weights is analogous to the factored Bellman error computation. For each branch of the policy, we generate a set of LP constraints (according to Spec. 6) that expresses the approximation error incurred by this branch. The union of all constraints is then

$$weight\_lp \coloneqq \textstyle\bigcup_{i<len(\pi)} branch\_lp(\pi_i, map(fst, \pi_{:i})).$$

Variables of the LP are the approximation error ϕ to minimize, and the weights w that determine the new weights. Given a set of LP constraints cs, $\langle cs \rangle_{LP}$ denotes the set of feasible solutions. We show that for any optimal solution $(\phi^*, w^*) \in \langle weight\_lp \rangle_{LP}$, $upd\_w \coloneqq w^*$ satisfies Spec. 1. We formally prove that the LP always has an optimal solution, as the set of potentially optimal solutions is compact.

Specification 6 ($branch\_lp$)**.** Given a partial state t, an action a, a list of partial states ts, $branch\_lp$ constructs an LP that minimizes the approximation error for the states $X_{(t,ts)}$:

$$\begin{aligned} (\phi, w) \in \langle branch\_lp(t, a, ts) \rangle_{LP} \longleftrightarrow \\ \forall x \in X_{(t,ts)}.\ \phi \geq |Q^a_w(x) - \nu_w(x)|. \end{aligned}$$

LPs for Branches For each branch of the policy, we proceed similarly to the Bellman error computation: we create two constraint sets, for positive and negative errors respectively (Alg. 6). We omit the scopes for brevity. At this level, we make use of another algorithm min_lp. Its first input C is a list of m scoped functions, the second input b is another list

Algorithm 6: Branch LP

$branch\_lp \coloneqq$
$min\_lp(C, -b \cdot \mathcal{I}') \cup min\_lp(-C, b \cdot \mathcal{I}')$
where $b \coloneqq map(inst_t, [R_i^a \mid i < r_a])$
$C \coloneqq map(inst_t, [h_i - \gamma g_i^a \mid i < m])$

of scoped functions. Now $min\_lp(C, b)$ creates an LP that minimizes $Cw - b$ w.r.t. w over all states (see Spec. 7). The definitions of C and b are analogous to the Bellman error computation. It then follows that $branch\_lp$ fulfills Spec. 6.

Specification 7 (min_lp)**.** $min\_lp(C, b)$ generates an LP that minimizes $Cw - b$ over all weights w:

$$(\phi, w) \in \langle min\_lp(C, b)\rangle_{LP} \longleftrightarrow$$
$$\forall x \in X.\ \phi \geq \textstyle\sum_{i<len(C)} w_i C_i(x) + \sum_{i<len(b)} b_i(x).$$

Factored LP Construction The algorithm min_lp resembles max_Σ, so we only point out the challenges encountered during verification. Full details can be found in the formalization. There are two significant modifications we made to the algorithm to make verification feasible. First, the original algorithm may create equality constraints that constrain variables to $-\infty$. Since these constraints are not supported by the LP solvers we use, we formally prove that one can modify the algorithm to omit such constraints without changing the set of feasible weights. Second, the combination of LP constraints in the definition of $weight\_lp$ requires some care, to avoid interactions between LP variables created in different branches. The min_lp algorithm creates new (private) LP variables and we need to make sure that these variables have distinct names for each branch. This issue was not discussed by Guestrin et al.. The problem can be solved by adding a tag to each generated variable. The tags contain t, a, and a flag to differentiate the two invocations of min_lp in each branch. For distinct tags p and p' we can then show that the solutions to the union of two constraint sets are equivalent to the intersection of the solution spaces of the individual constraint sets (concerning ϕ and w):

$$\langle min\_lp(p, C, b) \cup min\_lp(p', C', b')\rangle_{LP} =$$
$$\langle min\_lp(p, C, b)\rangle_{LP} \cap \langle min\_lp(p', C', b')\rangle_{LP}.$$

We show that min_lp creates an LP that is equivalent to the explicit (potentially exponentially larger) LP that has a constraint for each MDP state. It immediately follows that min_lp satisfies Spec. 7, which completes our correctness proof of API. With this approach to algorithm verification using loosely coupled locales, min_lp and max_Σ are not tied to MDPs and are thereby reusable components.

Code Generation

We now discuss the process of deriving a verified efficiently executable version from the verified abstract algorithm discussed above. To do so, we follow the methodology of program refinement (Wirth 1971), where one starts with an abstract, potentially non-executable version of the algorithm and verifies it. Then one devises more optimised versions of the algorithm, and only proves the optimizations correct in this latter step, thus separating mathematical reasoning from implementation specific reasoning. This approach was used in most successful algorithm verification efforts (Klein et al. 2009; Esparza et al. 2013; Kanav, Lammich, and Popescu 2014). In this work, the three most important stages are the initial abstract algorithm, an implementation with abstract data structures, and finally an implementation with concretized data structures. As a last step, we export verified code for API in the programming language Scala.

	Ring				Star	
n	$\mathbf{t(s)}$	$\mathbf{t_{LP}(s)}$	**Constrs**	**Vars**	$\mathbf{t(s)}$	$\mathbf{t_{LP}(s)}$
1	0.27	0.02	74	41	0.34	0.03
3	0.89	0.15	1258	693	0.50	0.05
5	1.89	0.46	4378	2455	0.69	0.08
7	3.78	0.98	9418	5305	0.98	0.14
9	6.74	1.82	16378	9243	1.30	0.22
11	12.44	3.36	25258	14269	1.52	0.29
13	20.95	5.31	36058	20383	1.76	0.36
15	34.69	8.67	48778	27585	2.28	0.50
17	58.30	16.86	63418	35875	2.89	0.64
19	92.19	30.25	79978	45253	3.69	0.80

Table 1: Evaluation on the ring and star domains. The first column denotes the number of clients. For each topology, the first two columns give the total running time and time spent in the LP solver. For the ring domain, we also show the number of LP constraints and variables generated.

Refinement using Locales Our implementation of stepwise refinement is based on Isabelle/HOL locales. For each locale of the abstract algorithm, we define a corresponding locale where we define the executable version of the algorithm. Finally, we relate the abstract version of the MDP to the concrete version. For each definition, we then show that corresponding inputs lead to corresponding outputs, i.e. our abstract algorithm and the implementation behave the same. At this point in the refinement, data structures remain abstract interfaces, with the concrete implementations chosen only later. We use the data structures provided by the Isabelle Collections Framework (Lammich and Lochbihler 2019) for code generation and we extend them with a data structure for scoped functions, represented as a pair of a function and a set for its scope. The data structure also provides an operation to evaluate a function on its full scope for memoization.

Certification of LP Solutions An implementation of API depends on efficient LP solvers. In our verified implementation, we use precise but unverified LP solvers and certify their results. This avoids implementing a verified, optimized LP solver but retains formal guarantees – the tradeoff here is that the unverified LP solver might return solutions that cannot be certified. At the cost of performance, it is also possible to connect the formalization to an existing simplex implementation for Isabelle/HOL (Spasić and Marić 2012). To achieve formal guarantees, the LP has to be solved exactly, i.e. using rational numbers. Two potential candidates for precise LP solvers are QSopt_ex (Applegate et al. 2007) and SoPlex (Bestuzheva et al. 2023). For larger LPs in our

setting, SoPlex demonstrated more consistent performance.

We certify optimality using the dual solution and the strong duality of linear programming. In our formalization we also formally prove that infeasibility and unboundedness can be certified similarly using farkas certificates or unbounded rays. The linear program is first preprocessed to a standard form: variable bounds and equality constraints are reduced to inequality constraints. The constraints of the resulting LP are of the form $Ax \leq b$, with no restrictions on x. During benchmarking of the certification process, the normalization operation usually applied to rationals after each operation proved to be very costly. For certification, we represent rational numbers as pairs that are never normalized, which leads to faster certificate checking in our experiments.

The exported Scala program takes an arbitrary function from linear programs to their solutions as input. If this function returns invalid solutions, they are rejected by the certificate checker, so there are no assumptions we need to place on the LP solver. However, there is the implicit assumption that the LP solver is deterministic. Since we are working in a fragment of a functional programming language, calling the LP solver twice on the same problem should lead to the same solution. In theory, a nondeterministic LP solver could be misused to lead to inconsistencies. As we do not compare LP solutions in our algorithm and SoPlex is actually deterministic, this problem does not impact our verified software. A more general solution to the problem could be the use of memoization or to model the nondeterminism with monads.

Experimental Evaluation We show the practicality of our verified implementation by applying it to both the ring and star topologies from (Guestrin et al. 2003). Note that all numerical computations have to be performed with infinite precision, which substantially impacts the performance. We run our implementation on an Intel i7-11800H CPU and set the discount factor to 0.9 in all our experiments. In all runs, the weights converged after max. 5 iterations. The results of the experiments (Table 1) show that the algorithm can deal with ring networks of half a million states and 20 actions. For larger networks, the precise mode of the LP solver SoPlex cannot find a rational solution. The experiment shows that our implementation of linear programming certification can process linear programs with tens of thousands of constraints. For the simpler star topology, we can handle systems with 2^{40} states in 45s.

Discussion

Our work combines and integrates a wide range of tools and formalization efforts to produce a verified implementation of API. In Isabelle/HOL, we build on the formal libraries for LPs (Thiemann 2022), MDPs (Hölzl 2017), linear algebra, analysis (Hölzl, Immler, and Huffman 2013), probability theory (Hölzl and Heller 2011), and the collections framework (Lammich and Lochbihler 2019). Furthermore, we certify the results computed by precise LP solvers. Our development consists of 20,000 lines of code, two-thirds focused on code generation. We show how to facilitate locales, powerful automation in Isabelle/HOL, and certification to develop complex formally verified software. We also make the case that the process of algorithm verification provides a detailed understanding of the algorithm: all hidden assumptions are made explicit, while we modularize the algorithm into components with precisely specified behaviour.

The methodology presented here can be applied to a large number of algorithms since the algorithm we verified is a seminal algorithm for solving factored MDPs, combining a large number of concepts that are widely used in many contexts, like AI (Delgado, Sanner, and De Barros 2011; Osband and Roy 2014; Deng, Devic, and Juba 2022) and model checking (Hinton et al. 2006; Dehnert et al. 2017).

Applications of interactive theorem provers in verification and formalizing mathematics have been recently attracting a lot of attention (Avigad and Harrison 2014; Avigad 2023; Massot 2021). In most applications, especially in computer science (Esparza et al. 2013; Klein et al. 2009) and AI (Bagnall and Stewart 2019; Selsam, Liang, and Dill 2017), the emphasis is on the difficulty of the proofs, whether that is due to many cases or complex constructions, etc., and how theorem proving helped find mistakes or find missing cases in the proofs. A distinct feature of this work is that its complexity comes from the large number of concepts it combines, shown in Fig. 2, which is a more prevalent issue in formalizing pure mathematics. Our project has contributed a better restructuring of the algorithm and untangling of the different concepts, leading to better understandability.

Multiple directions can be considered to extend and build on our work. An alternative to using exact LP solvers is to use their highly optimized floating point counterparts that do not calculate exact solutions. In this case, one needs to certify the error bound derived from a dual solution to the LP. Then the error bound has to be incorporated in the error analysis of the algorithm to obtain formal guarantees. Moreover, we may initialize the algorithm with weights computed by an unverified, floating point implementation of API to reduce number of iterations performed by the verified implementation. In Isabelle/HOL, some of the correctness proofs for code generation can be automated using tools to transfer theorems. Furthermore, the ergonomics for instantiating locales and inheriting from other locales could be improved for situations with many locale parameters.

Related Work Several formal treatments of MDPs have been developed in the theorem provers Isabelle/HOL (Hölzl 2017; Schäffeler and Abdulaziz 2023; Chevallier and Fleuriot 2021; Hartmanns, Kohlen, and Lammich 2023) and Coq (Vajjha et al. 2021). All developments verify algorithms for explicit MDPs, which limits their practical applicability to solve large MDPs. We adapt and integrate the implementation of Schäffeler and Abdulaziz with our formalization. The algorithm we verified in Isabelle/HOL was first presented in (Guestrin et al. 2003). An LP certification approach with Isabelle/HOL was previously done in the Flyspeck project (Obua and Nipkow 2009), where feasibility was checked. The work introduces a method using dual solutions from floating-point LP solvers to bound the objective value. The tool Marabou for neural network verfication uses Farkas vectors to produce proofs for its results (Isac et al. 2022).

Acknowledgements

This project was funded in part by the Deutsche Forschungsgemeinschaft – 378803395 (ConVeY).

References

Abdulaziz, M.; and Koller, L. 2022. Formal Semantics and Formally Verified Validation for Temporal Planning. In *The 36th AAAI Conference on Artificial Intelligence (AAAI)*.

Abdulaziz, M.; and Kurz, F. 2023. Formally Verified SAT-Based AI Planning. In *The 37th AAAI Conference on Artificial Intelligence (AAAI)*.

Abdulaziz, M.; and Lammich, P. 2018. A Formally Verified Validator for Classical Planning Problems and Solutions. In *The 30th International Conference on Tools with Artificial Intelligence (ICTAI)*.

Applegate, D. L.; Cook, W.; Dash, S.; and Espinoza, D. G. 2007. Exact Solutions to Linear Programming Problems. *Operations Research Letters*.

Avigad, J. 2023. Mathematics and the Formal Turn. arxiv:2311.00007.

Avigad, J.; and Harrison, J. 2014. Formally Verified Mathematics. *Commun. ACM*.

Bagnall, A.; and Stewart, G. 2019. Certifying the True Error: Machine Learning in Coq with Verified Generalization Guarantees. In *The 33rd AAAI Conference on Artificial Intelligence (AAAI)*.

Ballarin, C. 2014. Locales: A Module System for Mathematical Theories. *J. Autom. Reason.*

Bestuzheva, K.; Besançon, M.; Chen, W.-K.; Chmiela, A.; Donkiewicz, T.; van Doornmalen, J.; Eifler, L.; Gaul, O.; Gamrath, G.; Gleixner, A.; Gottwald, L.; Graczyk, C.; Halbig, K.; Hoen, A.; Hojny, C.; van der Hulst, R.; Koch, T.; Lübbecke, M.; Maher, S. J.; Matter, F.; Mühmer, E.; Müller, B.; Pfetsch, M. E.; Rehfeldt, D.; Schlein, S.; Schlösser, F.; Serrano, F.; Shinano, Y.; Sofranac, B.; Turner, M.; Vigerske, S.; Wegscheider, F.; Wellner, P.; Weninger, D.; and Witzig, J. 2023. Enabling Research through the SCIP Optimization Suite 8.0. *ACM Trans. Math. Softw.*

Budde, C. E.; Dehnert, C.; Hahn, E. M.; Hartmanns, A.; Junges, S.; and Turrini, A. 2017. JANI: Quantitative Model and Tool Interaction. In *The 23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems*.

Chevallier, M.; and Fleuriot, J. D. 2021. Formalising the Foundations of Discrete Reinforcement Learning in Isabelle/HOL. *CoRR*.

Dehnert, C.; Junges, S.; Katoen, J.-P.; and Volk, M. 2017. A Storm Is Coming: A Modern Probabilistic Model Checker. In *The 29th International Conference on Computer Aided Verification*.

Delgado, K. V.; Sanner, S.; and De Barros, L. N. 2011. Efficient Solutions to Factored MDPs with Imprecise Transition Probabilities. *Artificial Intelligence*.

Deng, Z.; Devic, S.; and Juba, B. 2022. Polynomial Time Reinforcement Learning in Factored State MDPs with Linear Value Functions. In *The 25th International Conference on Artificial Intelligence and Statistics*.

Esparza, J.; Lammich, P.; Neumann, R.; Nipkow, T.; Schimpf, A.; and Smaus, J.-G. 2013. A Fully Verified Executable LTL Model Checker. In *25th International Conference on Computer Aided Verification (CAV)*.

Guestrin, C.; Koller, D.; Parr, R.; and Venkataraman, S. 2003. Efficient Solution Algorithms for Factored MDPs. *J. Artif. Intell. Res.*

Hartmanns, A.; Junges, S.; Quatmann, T.; and Weininger, M. 2023. A Practitioner's Guide to MDP Model Checking Algorithms. In *TACAS (1)*, volume 13993 of *Lecture Notes in Computer Science*, 469–488. Springer.

Hartmanns, A.; Kohlen, B.; and Lammich, P. 2023. Fast Verified SCCs for Probabilistic Model Checking. In *The 21st International Symposium on Automated Technology for Verification and Analysis (ATVA)*.

Hinton, A.; Kwiatkowska, M. Z.; Norman, G.; and Parker, D. 2006. PRISM: A Tool for Automatic Verification of Probabilistic Systems. In *The 22nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*.

Hölzl, J. 2017. Markov Chains and Markov Decision Processes in Isabelle/HOL. *J. Autom. Reason.*

Hölzl, J.; and Heller, A. 2011. Three Chapters of Measure Theory in Isabelle/HOL. In *2nd International Conference on Interactive Theorem Proving (ITP)*.

Hölzl, J.; Immler, F.; and Huffman, B. 2013. Type Classes and Filters for Mathematical Analysis in Isabelle/HOL. In *4th International Conference on Interactive Theorem Proving (ITP)*.

Isac, O.; Barrett, C.; Zhang, M.; and Katz, G. 2022. Neural Network Verification with Proof Production. In *The 22nd Conference on Formal Methods in Computer-Aided Design (FMCAD)*.

Junges, S.; Jansen, N.; Katoen, J.; Topcu, U.; Zhang, R.; and Hayhoe, M. M. 2018. Model Checking for Safe Navigation Among Humans. In McIver, A.; and Horváth, A., eds., *Quantitative Evaluation of Systems - 15th International Conference, QEST 2018, Beijing, China, September 4-7, 2018, Proceedings*, volume 11024 of *Lecture Notes in Computer Science*, 207–222. Springer.

Kanav, S.; Lammich, P.; and Popescu, A. 2014. A Conference Management System with Verified Document Confidentiality. In *The 26th International Conference on Computer Aided Verification (CAV)*.

Keller, T.; and Eyerich, P. 2012. PROST: Probabilistic Planning Based on UCT. In McCluskey, L.; Williams, B. C.; Silva, J. R.; and Bonet, B., eds., *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012, Atibaia, São Paulo, Brazil, June 25-19, 2012*. AAAI.

Klein, G.; Elphinstone, K.; Heiser, G.; Andronick, J.; Cock, D.; Derrin, P.; Elkaduwe, D.; Engelhardt, K.; Kolanski, R.; Norrish, M.; Sewell, T.; Tuch, H.; and Winwood, S. 2009. seL4: Formal Verification of an OS Kernel. In *22nd ACM Symposium on Operating Systems Principles 2009 (SOSP)*.

Lahijanian, M.; Wasniewski, J.; Andersson, S. B.; and Belta, C. 2010. Motion planning and control from temporal logic specifications with probabilistic satisfaction guarantees. In *IEEE International Conference on Robotics and Automation, ICRA 2010, Anchorage, Alaska, USA, 3-7 May 2010*, 3227–3232. IEEE.

Lammich, P.; and Lochbihler, A. 2019. Automatic Refinement to Efficient Data Structures: A Comparison of Two Approaches. *J Autom Reasoning*.

Leroy, X. 2009. Formal Verification of a Realistic Compiler. *Commun. ACM*.

Massot, P. 2021. Why Formalize Mathematics?

Nipkow, T.; Paulson, L. C.; and Wenzel, M. 2002. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*.

Obua, S.; and Nipkow, T. 2009. Flyspeck II: The Basic Linear Programs. *Ann Math Artif Intell*.

Osband, I.; and Roy, B. V. 2014. Near-optimal reinforcement learning in factored MDPs. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'14, 604–612. Cambridge, MA, USA: MIT Press.

Sanner, S. 2010. Relational Dynamic Influence Diagram Language (RDDL): Language Description.

Schäffeler, M.; and Abdulaziz, M. 2023. Formally Verified Solution Methods for Infinite-Horizon Markov Decision Processes. In *The 37th AAAI Conference on Artificial Intelligence (AAAI)*.

Selsam, D.; Liang, P.; and Dill, D. L. 2017. Developing Bug-Free Machine Learning Systems With Formal Mathematics. In *The 34th International Conference on Machine Learning (ICML)*.

Spasić, M.; and Marić, F. 2012. Formalization of Incremental Simplex Algorithm by Stepwise Refinement. In *The 18th International Symposium on Formal Methods*.

Thiemann, R. 2022. Duality of Linear Programming. *Arch. Formal Proofs*.

Vajjha, K.; Shinnar, A.; Trager, B. M.; Pestun, V.; and Fulton, N. 2021. CertRL: Formalizing Convergence Proofs for Value and Policy Iteration in Coq. In *The 10th International Conference on Certified Programs and Proofs (CPP)*.

Wirth, N. 1971. Program Development by Stepwise Refinement. *Commun. ACM*.

Younes, H. L.; and Littman, M. L. 2004. PPDDL1. 0: The Language for the Probabilistic Part of IPC-4. In *Proc. International Planning Competition*.

Copyright and License Information

Association for the Advancement of Artificial Intelligence
1101 Pennsylvania Ave, NW, Suite 300
Washington, DC 20004 USA

AAAI COPYRIGHT FORM

Title of Article/Paper: Formally Verified Approximate Policy Iteration

Publication in Which Article/Paper Is to Appear: Proceedings of the 39th AAAI Conference on Artificial Intelligence (AAAI-25)

Author's Name(s): Maximilian Schäffeler, Mohammad Abdulaziz

Please type or print your name(s) as you wish it (them) to appear in print

PART A – COPYRIGHT TRANSFER FORM

The undersigned, desiring to publish the above article/paper in a publication of the Association for the Advancement of Artificial Intelligence, (AAAI), hereby transfer their copyrights in the above article/paper to the Association for the Advancement of Artificial Intelligence (AAAI), in order to deal with future requests for reprints, translations, anthologies, reproductions, excerpts, and other publications.

This grant will include, without limitation, the entire copyright in the article/paper in all countries of the world, including all renewals, extensions, and reversions thereof, whether such rights current exist or hereafter come into effect, and also the exclusive right to create electronic versions of the article/paper, to the extent that such right is not subsumed under copyright.

The undersigned warrants that they are the sole author and owner of the copyright in the above article/paper, except for those portions shown to be in quotations; that the article/paper is original throughout; and that the undersigned right to make the grants set forth above is complete and unencumbered.

If anyone brings any claim or action alleging facts that, if true, constitute a breach of any of the foregoing warranties, the undersigned will hold harmless and indemnify AAAI, their grantees, their licensees, and their distributors against any liability, whether under judgment, decree, or compromise, and any legal fees and expenses arising out of that claim or actions, and the undersigned will cooperate fully in any defense AAAI may make to such claim or action. Moreover, the undersigned agrees to cooperate in any claim or other action seeking to protect or enforce any right the undersigned has granted to AAAI in the article/paper. If any such claim or action fails because of facts that constitute a breach of any of the foregoing warranties, the undersigned agrees to reimburse whomever brings such claim or action for expenses and attorneys' fees incurred therein.

Returned Rights

In return for these rights, AAAI hereby grants to the above author(s), and the employer(s) for whom the work was performed, royalty-free permission to:

1. Retain all proprietary rights other than copyright (such as patent rights).
2. Personal reuse of all or portions of the above article/paper in other works of their own authorship. This does not include granting third-party requests for reprinting, republishing, or other types of reuse. AAAI must handle all such third-party requests.
3. Reproduce, or have reproduced, the above article/paper for the author's personal use, or for company use provided that AAAI copyright and the source are indicated, and that the copies are not used in a way that implies AAAI endorsement of a product or service of an employer, and that the copies per se are not offered for sale. The foregoing right shall not permit the posting of the article/paper in electronic or digital form on any computer network, except by the author or the author's employer, and then only on the author's or the employer's own web page or ftp site. Such web page or ftp site, in addition to the aforementioned requirements of this Paragraph, shall not post other AAAI copyrighted materials not of the author's or the employer's creation (including tables of contents with links to other papers) without AAAI's written permission.
4. Make limited distribution of all or portions of the above article/paper prior to publication.
5. In the case of work performed under a U.S. Government contract or grant, AAAI recognized that the U.S. Government has royalty-free permission to reproduce all or portions of the above Work, and to authorize others to do so, for official U.S. Government purposes only, if the contract or grant so requires.

In the event the above article/paper is not accepted and published by AAAI, or is withdrawn by the author(s) before acceptance by AAAI, this agreement becomes null and void.

(1) [redacted]

Author/Authorized Agent for Joint Author's Signature

12/16/2024

Date (mm/dd/yyyy)

Technische Universität München

Employer for whom work was performed

Title (if not author)

(For jointly authored Works, all joint authors should sign unless one of the authors has been duly authorized to act as agent for the others.)

AAAI copyright form

Re: Permission to Include AAAI Paper in Cumulative Thesis

Subject: Re: Permission to Include AAAI Paper in Cumulative Thesis
From: AAAI Proceedings Questions <proceedings-questions@aaai.org>
Date: 17.01.25, 07:01
To: Maximilian Schaeffeler <maximilian.schaeffeler@tum.de>

Dear Maiximilian,
You don't need any special permission to use your PDF in your dissertation. AAAI returns the right to you on this case.
Best regards,
Francisco Cruz,
AAAI Press Editorial Team

On Thu, Jan 16, 2025 at 3:35 PM Maximilian Schaeffeler <maximilian.schaeffeler@tum.de> wrote:

> Good Morning,
>
> I am seeking clarification regarding the use of articles authored by myself in a cumulative dissertation/doctoral thesis.
> Is this right granted as part of the AAAI copyright form or am I required to file a "Request to Reproduce Copyrighted Materials"?
> Specifically, I would need to include the full PDF of my 2023 AAAI paper "Formally Verified Solution Methods for Markov Decision Processes".
>
> Thank you for your assistance. I look forward to your clarification.
>
> Best regards,
>
> Maximilian Schäffeler
>
>

AAAI copyright clarification

Core Publication 3

Bram Kohlen, Maximilian Schäffeler, Mohammad Abdulaziz, Arnd Hartmanns, and Peter Lammich. "A Formally Verified IEEE 754 Floating-Point Implementation of Interval Iteration for MDPs." In: *Computer Aided Verification*. Lecture Notes in Computer Science. Springer, 2025. To appear.

A Formally Verified IEEE 754 Floating-Point Implementation of Interval Iteration for MDPs

Bram Kohlen[1], Maximilian Schäffeler[2], Mohammad Abdulaziz[2,3], Arnd Hartmanns[1], and Peter Lammich[1]

[1] University of Twente, The Netherlands b.kohlen@utwente.nl
[2] Technical University of Munich, Germany maximilian.schaeffeler@tum.de
[3] King's College London, United Kingdom

Abstract. We present an efficiently executable, formally verified implementation of interval iteration for MDPs. Our correctness proofs span the entire development from the high-level abstract semantics of MDPs to a low-level implementation in LLVM that is based on floating-point arithmetic. We use the Isabelle/HOL proof assistant to verify convergence of our abstract definition of interval iteration and employ step-wise refinement to derive an efficient implementation in LLVM code. To that end, we extend the Isabelle Refinement Framework with support for reasoning about floating-point arithmetic and directed rounding modes. We experimentally demonstrate that the verified implementation is competitive with state-of-the-art tools for MDPs, while providing formal guarantees on the correctness of the results.

1 Introduction

Probabilistic model checking (PMC) [3,4] is a formal verification technique for randomised systems and algorithms like wireless communication protocols [35], network-on-chip (NoC) architectures [47], and reliability and performance models [5]. Typical properties checked by means of PMC relate to *reachability probabilities*: What is the probability for a file to eventually be transmitted successfully [17]? Is the probability for a NoC router's queue to overflow within c clock cycles below 10^{-5}? What is the maintenance strategy that minimises service outages within a given cost budget [48,49]? The system models that PMC is applied to are specified in higher-level modelling languages such as Modest [11,23] or JANI [14] with a formal semantics in terms of (extensions of) Markov chains and Markov decision processes (MDPs) [10,45].

PMC delivers results with formal guarantees, typically that the computed and (unknown) true probabilities differ by at most a user-specified ε. PMC is thus well-suited for the design and evaluation of safety- and performance-critical systems. Over the past decade, however, we have witnessed several threats to the validity of PMC results. First and foremost, the most-used PMC algorithm, value iteration (VI), was shown to be *unsound*, i.e. produce arbitrarily wrong results for certain inputs [21]. Several sound replacements for VI were subsequently developed [22,29,46], yet their soundness proofs have so far been *pen-and-paper*

style with room for human error. For example, the pseudocode for the *sound VI* algorithm as stated in [46] contains a subtle omission that only surfaces on 1 of the 78 models of the Quantitative Verification Benchmark Set (QVBS) [30]. This calls for *formal specifications of the algorithms* accompanied by *machine-checked correctness proofs*. Even correct algorithms, however, may be incorrectly implemented in today's manually-coded PMC tools. As a case in point, the implementation of the *interval iteration* algorithm [22] for expected rewards [7] in the mcsta model checker of the Modest Toolset [27] diverges on some inputs. We thus need *correct-by-construction implementations*, too.

VI-based algorithms are iterative numeric approximation schemes that need to be implemented via fixed machine-precision floating-point arithmetic to obtain acceptable performance [15,28]. This introduces approximation and rounding errors that in turn may lead to incorrect Boolean outputs [57]. An efficient solution is to carefully use the directed rounding modes provided by standard IEEE 754 floating-point implementations as in all of today's common CPUs [26], which however needs careful *reasoning about floating-point errors and rounding* in all formal proofs and correctness-preserving implementation strategies.

Our contribution. We present a solution to all of the above challenges based on the interval iteration (II) algorithm [22] for sound PMC on MDP models and the interactive theorem prover (ITP) Isabelle/HOL [44] with its Isabelle Refinement Framework (IRF) [39]:

- We formalise (i.e. model) II in Isabelle/HOL's logic and formally prove its correctness using Isabelle/HOL (Sect. 3), making II the first sound PMC algorithm for MDPs with machine-checked correctness.
- We extend the IRF with support for floating-point arithmetic, including directed rounding modes (Sect. 4.2), making it the first ITP-based algorithm refinement approach suitable for II and similar algorithms.
- Using the IRF, we refine the formalisation of II into efficient LLVM bytecode (Sects. 4.3 and 4.4), delivering the first correct-by-construction implementation of a PMC algorithm.
- We embed the code into mcsta, a competitive probabilistic model checker (Sect. 5). We experimentally evaluate the performance using the QVBS, showing that the verified implementation is efficient. Our formal proofs and the benchmark code are available as part of the *additional files*.

State-of-the-art: Verification of Algorithms for MDPs. A probabilistic model checker like mcsta performs preprocessing and transformation steps for both correctness and performance. Previously, the strongly connected component [31] and maximal end component decomposition [32] algorithms have been verified down to LLVM, replacing their previous unverified implementations inside mcsta by verified ones of comparable performance. These were fully discrete graph algorithms, however, that neither required reasoning about numerical convergence in their correctness proofs nor floating-point arithmetic in their refinement to an efficient implementation. With this work, we contribute an essential piece for

the incremental replacement of unverified by verified algorithms for probabilistic reachability in mcsta's MDP model checking core.

Other work that is also relevant to our setting is the verification of iteration algorithms for MDPs: In Coq by Vajjha et. al. [55] and in Isabelle/HOL by Schäffeler and Abdulaziz [50,51]. In their work, they verified the classical version of value iteration and policy iteration, that optimise the expected discounted values, and a modified policy iteration algorithm for solving large, factored MDPs. We note that only Schäffeler and Abdulaziz [50,51] also verified practical implementations. However, since their implementations used infinite-precision arithmetic, they could not compete with state-of-the-art floating-point implementations. Thus, the work we present here is the first, up to our knowledge, where a full formal mathematical analysis of an algorithm, involving heavy usage of a formal mathematical probabilities library, is performed and a competitive floating-point implementation is also verified. Furthermore, from a formalisation-methodology perspective, we note that the correctness argument of II involves a substantial element of graph-theoretic reasoning, in addition to the reasoning about fixed points that is present in II and other verified MDP algorithms. This includes reasoning about connected components, acyclicity, and levels in a DAG, further complicating II's verification compared to other verified iteration algorithms.

State-of-the-art: Verification of Floating-Point Algorithms. That floating-point implementations deviate from the respective mathematical models of the algorithms is widely recognised as a problem. Bugs with potentially serious consequences were noted in the hardware and aerospace industry [25,43]. Due to the complexity of floating-point algorithms' behaviour, and the failure of testing to reliably catch bugs in those algorithms, there is a long tradition of applying formal methods to the verification of floating-point algorithms. This was done in verification systems like Z [9], HOL Light [24,25], PVS [13,42], and Coq [12,19]. Most of that previous work, however, focused on proving correctness of basic algorithms implemented in floating-point arithmetic. In contrast, we aim to do the correctness proofs on algorithms using real numbers, which we implement as floating-point numbers with directed rounding. This keeps our correctness proofs manageable while preserving interesting properties, even for complex programs.

A related line of work aims to prove correctness by providing error bounds. Tools like PRECiSA [54], FPTaylor [52], Real2Float [41], and Fluctuat [20] analyze the floating-point error propagation. They focus on determining the worst-case roundoff error. While more expressive than our approach, these tools have limited to no support for programs with complex control flow like the nested loops in the implementation of II.

The static analysis tool Astrée [16] is used in the aviation and automotive industry to check absence of runtime errors. While it supports programs using floating-point numbers, it cannot verify arbitrary correctness properties. Frama-C [36] has similar functionality but supports deductive verification. However, the properties supported are limited, e.g. that outputs lie in a given interval [37]. The situation is similar with other deductive verifiers like KeY [2], which can verify the absence of exceptional values like NaN and infinity [1].

2 Preliminaries

We now present the necessary background for the rest of the paper: we introduce Isabelle and the Isabelle Refinement Framework, followed by IEEE 754 floating-point numbers and Markov Decision Processes in Isabelle/HOL.

2.1 Isabelle/HOL

An *interactive theorem prover* (ITP) is a program that implements a formal mathematical system in which a user writes definitions and theorem statements, and constructs proofs from a set of axioms. To prove a theorem in an ITP, the user provides high-level steps, and the ITP fills in the details at the axiom-level.

We perform our formalization using the ITP Isabelle/HOL [44], which is a proof assistant for Higher-Order Logic (HOL). Roughly speaking, HOL can be seen as a combination of functional programming with logic. Isabelle is designed to be highly trustworthy: a small, trusted kernel implements the inference rules of the logic. Outside the kernel, a large set of tools implement proof automation and high-level concepts like algebraic data types. Bugs in these tools cannot lead to inconsistent theorems being proved, as the kernel refuses flawed proofs.

We aim to represent our formalization as faithfully as possible, but we have optimized the presentation for readability. The notation in Isabelle/HOL is similar to functional programming languages like ML or Haskell mixed with mathematical notation. Function application is written as juxtaposition: we write $f\ x_1\ \ldots\ x_n$ instead of the standard notation $f(x_1,\ \ldots\ ,x_n)$. Recursive functions are defined using the **fun** keyword and pattern matching. For partial functions, we use the notation $f = (\lambda x \in X.\ g\ x)$, to explicitly restrict the domain of the function to X. Where required, we annotate types as $x :: type$.

Isabelle/HOL provides a keyword **locale** to define a named context with assumptions, e.g. an MDP with well-formedness assumptions [8]. Locales can be interpreted and extended in different contexts, e.g. a locale for MDPs can be instantiated for a specific MDP, which yields all theorems from within that locale.

2.2 Isabelle Refinement Framework

Our verification of II spans the mathematical foundations of MDPs, the implementation of optimized algorithms and data structures, and the low-level LLVM intermediate language [40]. To keep the verification effort manageable, we use a stepwise refinement approach: starting with an abstract specification, we incrementally add implementation details, proving that each addition preserves correctness, e.g. computing a fixed-point by iteration, or implementing MDPs by a sparse-matrix data structure. The former specifies the control-flow of a program, but the datatype remains the same. The latter we call *data refinement*.

This approach is supported by the Isabelle Refinement Framework (IRF) [39]. In the IRF, we define algorithms in the *nondeterministic result (nres) monad*, where a program either fails or produces a set of results. The notation $a \leq_R c$ denotes that every (non-deterministic) output of abstract program a is related to

an output of concrete program c via the *refinement relation* R. In other words, c is an implementation of a. If a refinement step does not involve data refinement, then we use $\leq := \leq_{R_{id}}$ where R_{id} is the identity relation.

At the start of the refinement chain, we put our specification and at the end of the refinement chain, we aim to have an efficient LLVM program. Once sufficiently refined, the sepref [38] tool can automatically refine a program to LLVM. As $\leq_R$ is transitive, the LLVM program satisfies the specification.

2.3 Floating-Point Arithmetic

Our work uses the formalization of the IEEE 754 floating-point standard in Isabelle/HOL [58]. This library provides a generic type *(e,f) float* which resembles the scientific notation: e is the number of bits for the *exponent*, and f is the number of bits for the *fraction* (also known as mantissa).

We use the type *double* = *(11,52) float*. The floating-point format contains positive and negative numbers, as well as special values for $\pm\infty$ and *not a number (NaN)*. The function *valof :: (e,f) float* $\rightarrow$ *ereal* maps non-NaN floating-point numbers to *extended real numbers*, i.e., $\mathbb{R}\dot{\cup}\{-\infty, +\infty\}$. Finally, the formalization provides all standard floating-point instructions like addition, multiplication, and comparisons as well as intuitive predicates to identify special cases (e.g. *is_nan*).

2.4 Markov Decision Processes

Markov Decision Processes (MDPs) are widely used to model probabilistic systems with nondeterministic choices [45], e.g. in PMC, planning, operations research and reinforcement learning [6,53]. Intuitively, an *agent* interacts with an environment by choosing *actions* that, together with random elements, influence the *state* of the system. The agent has an *objective*, e.g. to reach certain states, and aims to choose actions that optimize the probability to achieve the objective. Many important concepts defined in this section are illustrated in Example 1.

Formally, a finite MDP is a pair $M = (S, K)$ where S is a finite, non-empty set of *states*, and $K : S \rightarrow 2^{\mathcal{P}(S)}$ is the transition kernel. It maps every state to a finite, non-empty set of *actions* in the form of transition probabilities. $\mathcal{P}(S)$ denotes the set of probability measures on S, i.e. functions $p : S \rightarrow [0, 1]$ where $\sum_{s \in S} p(s) = 1$. Furthermore for K is closed under S: Actions from S lead to S.

Our formalization of II extends the *Markov Models* [33,34] formalization from the Isabelle/HOL library. MDPs are modeled with a generic type *'s mdpc* and a locale *Finite_MDP* that, in combination, contain the states, the transition kernel and well-formedness conditions (Locale 2.1). In the following, we abbreviate the projections *states M* and *actions M* as S and K. The type of the states is *'s*, and the type of probability distributions over *'s* is *'s pmf*. For a *p :: 's pmf*, $set_{pmf}\ p$ denotes its support, i.e. the set of states with non-zero probability.

locale *Finite_MDP* = *(Locale 2.1)*
fixes $M :: 's\ mdpc$ **and** S **and** K
defines $S = states\ M$ **and** $K = actions\ M$
assumes $S \neq \varnothing$ **and** *finite* S **and** $\forall s.\ K\ s \neq \varnothing$ **and** $\forall s \in S.\ finite\ (K\ s)$
assumes $\forall s \in S.\ (\bigcup a \in K\ s.\ set_{pmf}\ a) \subseteq S$

Strategies choose an action based on the visited states. This formalization works with *configurations*, which are pairs of states and strategies. A configuration is *valid* if the strategy selects only enabled actions and the state of the configuration is in S. $valid_{cfg}$ denotes the set of all valid configurations. Given a configuration and an MDP, the probability space of infinite traces $T\ cfg$ is constructed from the induced Markov Chain, where each state is a configuration.

MDP subcomponents play an important role in the analysis of II. A *sub-MDP* $M' = (S', K')$ consists of a subset of states $S' \subseteq S$ and a restricted kernel K' where $\forall s \in S'.\ K'(s) \subseteq K(s)$. A sub-MDP is *strongly connected* if all states can reach each other via a sequence of actions. A closed, strongly connected sub-MDP is an *end component* (EC). A *maximal end component* (MEC) is an EC that is not a sub-MDP of another EC. Finally, *trivial MECs* are MECs with one state and no actions, and *bottom MECs* are MECs without an exit.

Reachability. In our PMC setting, the objective is to minimize or maximize the long-term reachability probabilities of a set of target states $U \subseteq S$. The value function $P_{cfg} :: 's \Rightarrow real$ gives the probability of reaching U in the Markov Chain induced by the configuration *cfg*. Minimal and maximal reachability probabilities are denoted by $P_{\inf}$ and $P_{\sup}$ respectively. $P_{\inf}$ is defined as the infimum of P_{cfg} over all valid configurations, $P_{\sup}$ is defined using the supremum. We also introduce the *Bellman optimality operators* $F_{\inf}$ and $F_{\sup}$ (Def. 2.1, $F_{\sup}$ omitted). For a state $s \in S$ and a value vector v, $F_{\inf}\ v\ s$ denotes the minimal expected value of x after one step from s. The symbol $\bigsqcap$ denotes the infimum.

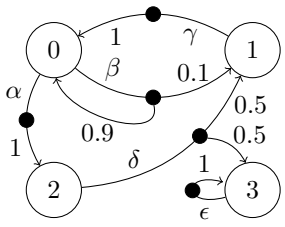


Fig. 1. A simple MDP with four states and five actions.

definition $F_{\inf}\ v = (\lambda s \in S.\ \textbf{if}\ s \in U\ \textbf{then}\ 1\ \textbf{else}$ *(Def. 2.1)*
$\bigsqcap a \in K\ s.\ \sum t \in set_{pmf}\ a.\ v\ t * pmf\ a\ t)$

The *least fixed point* (*lfp*) of $F_{\inf}$ is $P_{\inf}$. In other words, repeatedly applying $F_{\inf}$ to a lower bound of $P_{\inf}$ converges to $P_{\inf}$ in the limit. For II, we preprocess the MDP such that the *greatest fixed point* (*gfp*) of $F_{\inf}$ is the same as $P_{\inf}$, and then iterate $F_{\inf}$ on both a lower and an upper bound until they closely approximate $P_{\inf}$. The same holds for $F_{\sup}$ and $P_{\sup}$.

Example 1. Fig. 1 shows an MDP with four states, $S = \{0, 1, 2, 3\}$. The outgoing transitions from each state represent the actions in the MDP. Each transition leads to a black dot and branches into the successor states with corresponding probabilities. For example in state 0, $K(0) = \{\alpha, \beta\}$. The agent can choose α to move to state 2, or β to have a 10% chance to move to state 1.

Let the target states $U = \{3\}$. The reachability probabilities are $P_{\text{inf}}(2) = 0.5$ and $P_{\text{sup}}(2) = 1$. The MDP has a single bottom MEC $\{3\}$ with action $\{\epsilon\}$, and a single trivial MEC $\{2\}$. The states $\{0, 1\}$ form a MEC with actions β and γ.

3 Interval Iteration in Isabelle/HOL

The *interval iteration (II)* algorithm for MDPs is an iterative solution method for reachability problems based on value iteration. In contrast to standard value iteration, there is a simple and sound stopping criterion. We present our formalization of definitions and correctness proofs in Isabelle/HOL of II and preprocessing routines. Our formalization is based on the proofs in [22], we highlight the challenges encountered during formalization and point out differences in our formal proofs. In particular, we present a more elegant and much more precise proof of [22, Proposition 3]. Moreover, we simplify the definitions of the preprocessing steps significantly. In the following, all statements prefixed with **lemma** or **theorem** are formally verified in Isabelle/HOL. All theorems and definitions for P_{sup} that are analogous to the ones for P_{inf} are omitted here for brevity.

3.1 The Interval Iteration Algorithm

The idea of the II algorithm is to start with a lower and an upper bound on the true reachability probability and iterate the Bellman optimality operator F_{inf} (F_{sup}) on both. Since the optimality operators are monotone, both sequences converge to a fixed point. On arbitrary MDPs, these fixed points are not necessarily the same. However, if the MDP is preprocessed to only contain MECs that are trivial or bottom MECs, both fixed points are equal to the optimal reachability probabilities.

For now, assume an arbitrary MDP with a single target state s_+ and an avoid state s_-, that are both sinks. As the initial lower bound lb_0, we take the function that assigns 1 to s_+ and 0 to all other states. The initial upper bound ub_0 assigns 0 to s_- and 1 to all other states. The II algorithm computes the sequences $lb_{\text{inf}}\ n$ and $ub_{\text{inf}}\ n$, defined as the n-fold application of F_{inf} to lb_0 and ub_0:

definition $lb_{\text{inf}}\ n = (F_{\text{inf}})^n\ lb_0$ **and** $ub_{\text{inf}}\ n = (F_{\text{inf}})^n\ ub_0$ *(Def. 3.1)*

It is an immediate consequence of the monotonicity of the Bellman optimality operators that the lower (upper) bounds are monotonically increasing (decreasing). Clearly, lb_0 is a lower bound and ub_0 is an upper bound of P_{inf}. Additionally, we formally derive that the Bellman optimality operators preserve upper and lower bounds in Lemma 3.1. These two properties of the abstract II algorithm are required for the refinement proof in Sect. 4.

lemma **assumes** $v \leq P_{\text{inf}}$ **shows** $F_{\text{inf}}\ v \leq P_{\text{inf}}$ *(Lemma 3.1)*
lemma **assumes** $v \geq P_{\text{inf}}$ **shows** $F_{\text{inf}}\ v \geq P_{\text{inf}}$

3.2 Reduced MDPs

II is only guaranteed to converge if the MDP only contains trivial or bottom MECs. We therefore need to preprocess the MDP before applying II. The preprocessing steps differ for P_{inf} and P_{sup}, they are called *min-reduction* and *max-reduction*. In a first step, we extend the existing MDP formalization [33] with *strongly connected components* (SCCs) and bottom MECs (Def. 3.2). The states of an MDP that form trivial or bottom MECs are called *trivials* or *bottoms* respectively. We follow [22] and call an MDP *reduced* if all of its MECs are either trivial or bottom MECs.

definition *bmec M b* = *(Def. 3.2)*
mec M b $\wedge$ ($\forall s \in$ *states b. actions b s* = *K s*)

Min-Reduction. The min-reduction for MDPs transforms an arbitrary MDP into a reduced MDP with the same P_{inf}. The idea is that all non-trivial MECs (except s_+) can reach the target with probability 0: There exists a strategy that stays in the MEC forever and therefore never reaches s_+. Hence, all such MECs may be collapsed into a single absorbing state s_-. To formalize this transformation, we first define a function red_{inf} (Def. 3.3) mapping the states, and then apply it to the MDP M to obtain the reduced MDP M_{inf} (Def. 3.4). Our formal definition is a substantial simplification compared to [22, Def. 4].

definition $red_{\text{inf}}\ s$ = **if** $s \in$ *trivials* $M \cup \{s_+\}$ **then** s **else** s_- *(Def. 3.3)*

definition M_{inf} = *fix_loop* s_- ($map_{mdpc}\ red_{\text{inf}}\ M$) *(Def. 3.4)*

The function map_{mdpc} (defined in [34]) applies a function to every state of an MDP. If the function merges states, map_{mdpc} merges the action sets. Finally, *fix_loop* s_- replaces the actions at s_- with a single self-loop. Fig. 2 displays the min-reduced version of the MDP from Fig. 1.

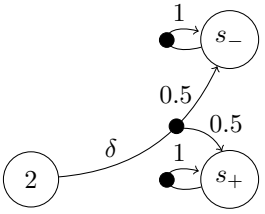


Fig. 2. Min-reduced version of the MDP in Fig. 1.

The formal proof of the fact that M_{inf} is in fact a reduced MDP follows the original [21]. We also show that the transformation preserves P_{inf}. To distinguish the reachability probabilities of the original and the reduced MDP, we use the notation P_{inf} for the original MDP and $M_{\text{inf}}.P_{\text{inf}}$ for the reduced MDP. Our proof is based on the fact that min-reduction preserves the finite-horizon probabilities $P^{\leq}_{\text{inf}}\ n$ (Lemma 3.2), i.e. the reachability probability in n steps. Now, the main claim (Thm. 3.1, [22, Proposition 3]) is a direct consequence. Note that our proof is simpler and more precise than the original: in [22], the authors only claim without details that for every strategy in the reduced MDP, there exists a strategy in the original MDP with the same P_{inf} and vice versa.

lemma **assumes** $s \in S$ **shows** $P^{\leq}_{\text{inf}}\ n\ s = M_{\text{inf}}.P^{\leq}_{\text{inf}}\ n\ (red_{\text{inf}}\ s)$ *(Lemma 3.2)*

theorem **assumes** $s \in S$ **shows** $P_{\text{inf}}\ s = M_{\text{inf}}.P_{\text{inf}}\ (red_{\text{inf}}\ s)$ *(Thm. 3.1)*

Max-Reduction. The P_{sup} case can be handled similarly with a max-reduction. The procedure is more involved, as not all non-trivial MECs can be collapsed while preserving P_{sup}: A maximizing strategy might choose to leave a non-trivial MEC. We can, however, first collapse each MEC into a single state to obtain an MDP M_{MEC}. In a second step, we map the bottom MECs to s_-. Finally, we remove self-loops at all states except s_+ and s_-. Our formalization decomposes max-reduction [22, Def. 5] into multiple steps. This again simplifies both the definitions and proofs.

Collapsing the MECs into a single state is done by the function *the_mec*, the transformation preserves P_{sup} (Thm. 3.2). Our proof resembles the proof of [18, Theorem 3.8], however we have to work around the fact that the MDP formalization [33] only supports deterministic policies. Note that every state of M_{MEC} now forms its own MEC. The correctness proof of the second phase of the reduction is similar to min-reduction. See Fig. 3 for the max-reduced version of the MDP from Fig. 1.

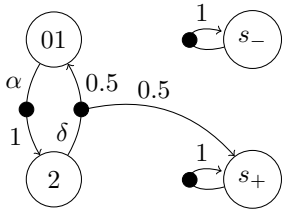


Fig. 3. Max-reduced version of the MDP in Fig. 1.

theorem assumes $s \in S$ *(Thm. 3.2)*
shows $P_{\text{sup}}\ s = M_{MEC}.P_{\text{sup}}\ (\textit{the\_mec}\ M\ s)$

Proof. ($\leq$) As collapsing MECs only shortens paths in the MDP, the proof proceeds similarly to the one for min-reduction via finite-horizon probabilities.

($\geq$) Consider an optimal strategy π_{MEC} in M_{MEC}, we need to show that there exists a strategy in M with the same reachability probability. Every MEC m of M contains a state s_m^π where the action selected by π_{MEC} is enabled. Moreover, within a MEC, we can obtain a deterministic, memoryless strategy π_m that reaches this state with probability 1. Thus we can construct a strategy in M that behaves like π_m within each MEC until s_m^π is reached and then follows π_{MEC}. The reachability probability of this strategy in M is the same as in M_{MEC}.

3.3 Reachability in Reduced MDPs

From now on we assume that we are working with a reduced, finite MDP M where each state is a trivial or bottom MEC. We show that in such an MDP, over time any strategy reaches a bottom MEC almost surely. This is the key property that will then allow us to prove the convergence of II.

Level Graph. First, we build a level graph of the MDP, starting at the bottom MECs (Def. 3.5). At level $n+1$, we add all those states where every action has a successor in level n or below. We define I to be the greatest non-empty level of the level graph G, so I is the number of steps that allows us to reach a bottom MEC from every state. We formally show that G has the desired properties, i.e. it is acyclic and contains every state at exactly one level. The proofs in Isabelle/HOL require substantial reasoning about graph-theoretic properties, e.g. we need to show that every MDP contains a bottom MEC.

fun G **where** *(Def. 3.5)*
$G\ 0 = \mathit{bottoms}\ M$
$G\ (n+1) =$ **let** $G_{\leq n} = \bigcup i \leq n.\ G\ i$ **in**
$\{s \in S \setminus G_{\leq n}.\ \forall a \in K\ s.\ G_{\leq n} \cap a \neq \varnothing\}$

Reachability of BMECs. We now show that intuitively, every strategy eventually descends through the levels of G. The rate at which a bottom MEC is encountered depends on η, the smallest probability of any transition in the MDP. At every step, the probability of descending a level wrt. G is at least η. Hence we can show that for any valid configuration, the probability to reach the bottom MECs in I steps is no less than η^I:

lemma **assumes** $\mathit{cfg} \in \mathit{valid}_{\mathit{cfg}}$ **shows** $\eta^I \leq P^{\leq}_{\mathit{cfg}}\ I$ *(Lemma 3.3)*

The value $P^{\leq}_{\mathit{cfg}}\ n$ denotes the finite-horizon reachability probability of the bottom MECs in n steps under configuration *cfg*. Note that the lemma was originally stated for safety instead of reachability problems. We transform it using the well-known equivalence $\mathbb{P}^{\leq n}_{\vec{\pi}}(\Diamond U) = 1 - \mathbb{P}^{\leq n}_{\vec{\pi}}(\Box \neg U)$. For multiples n of I, we obtain a stronger lower bound of $1 - (1-\eta^I)^n$ (Thm. 3.3, [22, Proposition 1]). As n increases, $(1-\eta^I)^n$ converges to 0 and thus $P^{\leq}_{\mathit{cfg}}\ nI$ tends towards 1. Since we chose *cfg* arbitrarily, we almost surely reach a bottom MEC in the limit.

theorem **assumes** $\mathit{cfg} \in \mathit{valid}_{\mathit{cfg}}$ **shows** $1 - (1-\eta^I)^n \leq P^{\leq}_{\mathit{cfg}}\ nI$ *(Thm. 3.3)*

3.4 Convergence of Interval Iteration

We assume a special form of reduced MDPs, where the only bottom MECs are the target state $\{s_+\}$ and avoid state $\{s_-\}$ that both are absorbing (Locale 3.1). The reduced MDPs from Sect. 3.2 are instances of this locale.

locale *MDP_Reach* = *Finite_MDP M* + *(Locale 3.1)*
assumes $s_- \in S$ **and** $s_+ \in S$ **and**
$\forall s \in S \setminus \{s_+, s_-\}.\ s \in \mathit{trivials}\ M$ **and**
$K\ s_- = \{\mathit{return}_{\mathit{pmf}}\ s_-\}$ **and** $K\ s_+ = \{\mathit{return}_{\mathit{pmf}}\ s_+\}$

Towards a convergence proof of II, we show that the lower and upper bound sequences relate to finite-horizon reachability probabilities of both s_+ and s_- (Lemma 3.4, [22, Lemma 4]). In this section, we indicate the target sets explicitly.

lemma **assumes** $s \in S$ *(Lemma 3.4)*
shows $\mathit{lb}_{\mathrm{inf}}\ n\ s = P^{\leq}_{\mathrm{inf}}\ \{s_+\}\ n\ s$ **and** $\mathit{ub}_{\mathrm{inf}}\ n\ s = 1 - P^{\leq}_{\mathrm{sup}}\ \{s_-\}\ n\ s$

With Thm. 3.3 we can bound the distance between the sequences (Thm. 3.4). Note that convergence is in general only guaranteed if all computations are carried out with arbitrary precision arithmetic. In a floating-point setting, the convergence to a unique fixed point is not guaranteed. Still, this theoretical result motivates the usage of the II algorithm to optimally solve reachability problems

on MDPs. In practice, on most instances the algorithm converges much faster than the theoretical bound suggests (see Sect. 5 for experimental results).

Finally, like [22], the theorem does not apply if all probabilities in the MDP are equal to one, i.e. no branching after an action is selected. In this case, the MDP is deterministic and is better solved with qualitative solution methods.

theorem *(Thm. 3.4)*
assumes $s \in S$ **and** $\epsilon > 0$ **and** $\eta \neq 1$ **and** $n \geq \lceil \log_{(1-\eta^I)} \epsilon \rceil * I$
shows $ub_{\text{inf}}\ n\ s - lb_{\text{inf}}\ n\ s \leq \epsilon$

Proof. As a first step, we show for all i:

$$\begin{aligned}
&ub_{\text{inf}}\ iI\ s - lb_{\text{inf}}\ iI\ s = 1 - P^{\leq}_{\text{sup}}\ \{s_-\}\ iI\ s - P^{\leq}_{\text{inf}}\ \{s_+\}\ iI\ s && \textit{(Lemma 3.4)}\\
&\leq 1 - (P^{\leq}_{\text{inf}}\ \{s_-\}\ iI\ s + P^{\leq}_{\text{inf}}\ \{s_+\}\ iI\ s) && (P^{\leq}_{\text{inf}} \leq P^{\leq}_{\text{sup}})\\
&= 1 - P^{\leq}_{\text{inf}}\ \{s_-, s_+\}\ iI\ s && \textit{(Disjoint events)}\\
&\leq (1 - \eta^I)^i && \textit{(Thm. 3.3)}
\end{aligned}$$

Set $i = \lceil \log_{(1-\eta^I)} \epsilon \rceil$ and the theorem follows from monotonicity.

4 Refinement using Floating-Point Arithmetic

In the next step, we use the IRF to refine the abstract specification of II to an efficient LLVM implementation. In our executable version, we implement real numbers using IEEE 754 double precision floating-point numbers (floats). Due to dedicated hardware support on modern consumer processors, floats have superior performance compared to any other decimal format. However, the rounding errors inherent to floating-point arithmetic make the refinement tricky. We propose an approach based on directed rounding modes to refine reals to *upper bounding* or *lower bounding* floats. A further challenge is that during II, the rounding mode needs to be switched regularly. Most consumer CPUs set the rounding mode through a global flag, which is both time-consuming at runtime [26] and cumbersome to reason about in the IRF. To circumvent this, we use the AVX512 instruction set which supports operation-specific rounding modes.

We first introduce the sepref tool for refinement to LLVM, after which we present our extension of the IRF and sepref for floats. We use that to obtain correct-by-construction LLVM code for II.

4.1 The Sepref Tool

We use sepref [38] to automatically refine an algorithm to an LLVM program. The sepref tool provides a library of verified, reusable data structures. As these data structures may access the *heap*, we need to use *(separation logic) assertions* that extend refinement relations with a heap.

Example 2. We demonstrate how to automatically refine to LLVM using sepref through the following example.

```
1 definition ls_app x xs = xs @ [x] and half_nat n = n div 2
2 definition apphalf n xs = do { let n' = half_nat n; return ls_append n' xs }
3 lemma (rshift1, half_nat) :: A_size → A_size
4 lemma (arl_app, ls_app) :: [λn xs. length xs < 2^63−1] A_size → A^d_arl → A_arl
5 lemma (arl_apphalf, apphalf) :: [λn xs. length xs < 2^63−1] A_size → A^d_arl → A_arl
```

Line 1 defines *ls_app* (insert an element at the end of a list) and *half_nat* (divide a natural number in half). Both are used in the definition of *apphalf* in Line 2. The standard library of sepref has LLVM implementations for lists as array lists (A_{arl}), and natural numbers as 64-bit signed words (A_{size}). The A indicates that these are assertions. For example, *half_nat* can be refined with the LLVM program *rshift1*, which performs an efficient bit shift to the right.

For sepref to use such refinements, they need to be in *parametric functor notation* as in Line 3. The lemma states that *rshift1* and *half_nat* are related as follows: if the inputs of *half_nat* (type *nat*) and *rshift1* (type *size*) are related via A_{size}, then the outputs are related via A_{size}. Line 4 provides a refinement of the append operation following the same principle, only now with two inputs. Moreover, the precondition $\mathit{length}\ xs < 2^{63} - 1$ limits the length of the input list. Finally, the superscript ^{d} indicates that the input is destructively updated. With all these rules registered to sepref, we can automatically refine *app_half*. We provide a signature so that sepref knows which data structure to use:

$$[\lambda n\ xs.\ \mathit{length}\ xs < 2^{63} - 1]\, A_{size} \to A^d_{arl} \to A_{arl}.$$

sepref automatically translates this into the LLVM program *arl_app_half* based on *rshift1* and *arl_append*. We also obtain the refinement relation in Line 5.

4.2 Floating-Point Extension of the Isabelle Refinement Framework

We extend the IRF with two data refinements to reason about floating-point arithmetic: real numbers to lower bounding (*lb*) or upper bounding (*ub*) floats. Since the *ub* case is mostly symmetric to the *lb* case, we focus on *lb* floats. We aim to construct a refinement relation that never produces NaN for the operations we support. NaN is incomparable, rendering it incompatible with a framework that reasons about bounds. Furthermore, the operations we support must preserve upper/lower bounds: the float -2_f (subscript f denotes floats) is a lower bound of 1, yet $-2_f * -2_f = 4_f$ is not a lower bound of $1 * 1 = 1$.

To resolve this, we only consider non-negative floats. We define the refinement relation $R_{lb} = \{(\mathit{fl},r).\ \mathit{valof}\ \mathit{fl} \leq r \wedge \neg \mathit{is\_nan}\ \mathit{fl} \wedge \mathit{valof}\ \mathit{fl} \geq 0\}$ which relates reals to *lb* floats, e.g. $(2_f, 3) \in R_{lb}$, but $(-2_f, -1) \notin R_{lb}$, as -2_f is negative. Since floats are *pure*, e.g. they do not need to allocate memory on the heap. This means that the assertion A_{lb} ignores the heap and is in essence R_{lb}.

We now present a non-exhaustive list of operations supported by our framework. We focus on the operations required for our use-case.

Fused multiply-add. The ternary operation *fma a b c* = $a * b + c$ (fused multiply-add) yields a smaller floating-point rounding error compared to separately multiplying and then adding. We name the AVX512 operation for *fma* with rounding mode *to_negative_infinity fma_avx_lb* and prove the following refinement:

lemma (*fma_avx_lb, fma*) :: $A_{lb} \to A_{lb}^{>0} \to A_{lb} \to A_{lb}$ *(Lemma 4.1)*

This lemma says that if the inputs are *lb*, not NaN and non-negative, the output is also *lb*, not NaN and non-negative. Note that the result of $0_f * \infty_f$ is NaN. We resolve this with the stricter assertion $A_{lb}^{>0}$ that only allows positive finite floats. In the case of II, these are the transition probabilities from the input model.

Comparison. Comparisons are not preserved among equally bounding floats since floating-point errors can stack up arbitrarily. We can only preserve information partially by implementing them as mixed operations (e.g. comparing *lb* to *ub*).

definition *leq_sound a b* = **spec** ($\lambda r.\ r \longrightarrow a \leq b$) *(Lemma 4.2)*
lemma (*leq_double, leq_sound*) :: $A_{ub} \to A_{lb} \to A_{bool}$

We use **spec** to introduce non-determinism as follows: the operation must return *False* if $a > b$ and can return anything otherwise. Consider the following 2 cases. Case 1: 4_f is a *ub* of 2, and 3_f is an *lb* of 5, we have $2 \leq 5$ but $4_f \not\leq 3_f$; Case 2: 3_f is a *ub* of 2, and 4_f is an *lb* of 5, we have $2 \leq 5$ and $3_f \leq 4_f$. So a single comparisons of reals has two valid outcomes in our refinement relation. Similarly, by swapping rounding modes we get the converse specification.

We implement subtraction as a mixed operator for similar reasons (omitted).

Min and max. It is possible to refine the minimum (*min*) and maximum (*max*) operations directly using comparisons. We define the following refinement:

definition *min_double fl1 fl2* = **if** *fl1* $\leq$ *fl2* **then** *fl1* **else** *fl2* *(Lemma 4.3)*
lemma (*min_double, min*) :: $A_{lb} \to A_{lb} \to A_{lb}$

This refinement holds despite the fact that a comparison does not reveal anything about the bounding float number. Consider the following case: 4_f is a lower bound of 5 and 3_f is a lower bound of 6. *min_double* 4_f 3_f = 3_f is a lower bound of *min 5 6 = 5*, even though the floating-point implementation returns the first argument, while the definition on reals returns the second argument. The refinement works analogously for *ub* with *min* and *lb*/*ub* with *max*.

Constants. We provide the obvious refinements for the real number constants 0 and 1, which can be exactly represented as floating-point numbers.

4.3 Refinement of Interval Iteration

Using our floating-point extension to the IRF, we derive an implementation of II of the abstract specification from Sect. 3 using floats and conservative rounding. The IRF allows us to reuse the correctness proofs of the abstract specification,

and reason about the correctness of the implementation in isolation. Through this separation of concerns we avoid directly proving the floating-point implementation correct.

The plot in Fig. 4 shows a fictive run of II on both reals and their refinement to bounding floats. In the long run, II converges to the dashed red line P_{inf}. The green lines denote the valuations of an MDP state using reals, lb starting from lb_0 and ub from ub_0. Implementing lb with floats using A_{lb} yields the violet line lb_f (similarly for ub using A_{ub}). Note that the deviations are exaggerated in this example. In practice, the errors are so small that no visual differences would appear in a to-scale plot. Formally, the following specification states soundness of II, i.e. the outputs are lower and upper bounds of the reachability probability:

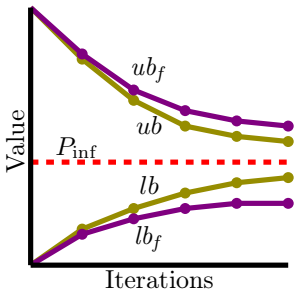


Fig. 4. The valuation of an MDP state over successive iterations: reals (green) vs. floats with safe rounding (violet). The dashed line marks the reachability probability.

definition *ii_inf_spec M* = *(Def. 4.1)*
  spec $(\lambda(x,\ y).\ \forall s \in \mathit{states}\ M.\ x\ s \le P_{\text{inf}}\ M\ s \wedge P_{\text{inf}}\ M\ s \le y\ s)$

Despite the fact that convergence follows from Thm. 3.4, we have excluded it from the specification. As we will discuss in Sect. 4.5, this would yield a void statement for our implementation with floats. As a first step towards the refinement to LLVM, we define II in the nres-monad (the *sup* case is analogous):

```
1  definition ii_gs_inf M L =                                   (Def. 4.2)
2    x ← lb_0 M; y ← ub_0 M; i ← 0; flag ← True;
3    while (i++ < L ∧ flag) (
4      (x,y) ← F^gs_inf M x y
5      flag ← spec(λx. True))
6    return (x,y)
```

We define *ii_gs_inf* in Line 1. It takes as inputs an MDP M, and a maximal iteration count L to guarantee termination. Line 2 initializes variables, most importantly the lower bounds lb and upper bounds ub. The *flag* nondeterministically decides whether to abort the loop. This is sound, because *ii_inf_spec* is satisfied after any number of iterations. In each iteration, we first update the valuations according to a Gauss-Seidel variant of F_{inf} (Line 4): we update lb and ub in-place, thereby we already use the updated values in the current iteration and converge faster.

The algorithm is now in a format ready for refinement proofs to LLVM. Using the setup from Sect. 3 and Lemma 3.1, it is straightforward to prove that the algorithm refines the specification:

theorem *ii_gs_inf M L* $\le$ *ii_inf_spec M* *(Thm. 4.1)*

4.4 Refinement of the mcsta Data Structure

The motivation behind refining II to LLVM code is to embed it into the model checker mcsta from the MODEST TOOLSET [27]. mcsta is an explicit-state probabilistic model checker that also supports quantitative model checking of MDPs. To avoid costly conversions of the MDP representation at runtime, we refine the *mdpc* data structure to the MDP data structure of mcsta. This is a two-step process: First, we do a data refinement from *mdpc* to the sparse-matrix representation used by mcsta based on HOL lists. Then, we use sepref to refine this data structure to LLVM. The sparse-matrix representation that we use is a 6-tuple:

(*St::nat list, Tr::nat list, Br::nat list, Pr::real list,* u_a*::nat,* u_t*::nat*).

For each state, *St* contains an index to *Tr*, pointing to the first transition (action) of the state. Similarly, for each transition, *Tr* contains its first index *Br* and *Pr*, pointing to the first branch of the transition and its probability. Finally, *Br* contains the target of the branch, pointing to *St*. Additionally, u_a and u_t are the avoid and target states respectively. Example 3 illustrates this data structure.

Example 3. A possible representation of the MDP from Fig. 1 is *St* = [*0,2,3,4,5*], *Tr* = [*0,1,3,4,6,7*], *Br* = [*2,0,1,0,1,3,3*], *Pr* = [*1.0,0.9,0.1,1,0.5,0.5,1.0*]. The index of the avoid and target state are stored in u_a and u_t. State 0 has two actions: α and β (transitions 0 and 1). Transition 1 (action β) has two branches, 1 and 2, that lead to state 0 and 1 with probability 0.9 and 0.1 respectively.

Refinement Relation. We relate the abstract MDP type *mdpc* to the concrete data structure of mcsta with the refinement relation R_M (definition omitted). For example, R_M contains a tuple of the MDP of Fig. 1 and Example 3 as well as with each other instance of the data structure in Sect. 4.4 along with the MDP it represents. These lists present in the Isabelle/HOL model of the data structure can be directly refined to arrays of 64-bit integers (signed for compatibility with mcsta). Through composition we obtain assertion A_M that maps an abstract MDP to LLVM.

Refinement of Operations. We use sepref to refine the functions lb_0, ub_0, F^{gs}_{inf} and *flag* that are used by II. Refining lb_0 and ub_0 to the concrete data structure is straightforward: we initialize an array and set entries to constants 0_f or 1_f. The floating-point refinement of F^{gs}_{inf} builds on *fma* and *min* (*max* for F^{gs}_{sup}) from Sect. 4.2. Finally, we implement *flag* as follows: we compare the upper and lower bound, and set the flag if the difference is less than ε, specified by the user. Using the above refinements for all operations in *ii_gs_inf*, we use sepref to obtain an LLVM algorithm *ii_gs_inf_llvm* within Isabelle/HOL.

4.5 Correctness Statement

For the final step in our proof, we combine all of our proofs into one theorem. We use the Hoare-triple because it allows is to write a concise correctness statement

of a program while blending out all of the intermediary tools like the parametric functor notation. We show the correctness of program *ii_gs_inf_llvm* against the specification *ii_inf_spec*. The resulting triple combines Thm. 4.1 with the refinements from Sect. 4.4 through transitivity.

1 **theorem** *llvm_htriple* *(Thm. 4.2)*
2 $(A_{size}\ n\ n_i \star A_{size}\ L\ L_i \star A_{ub}\ \varepsilon\ \varepsilon_f \star A_M\ M\ M_i$
3 $\star \uparrow(MDP\_Reach\ M \wedge n{+}1 < max\_size \wedge n = card\ (states\ M)))$
4 $(ii\_gs\_inf\_llvm\ L_i\ n_i\ \varepsilon_f\ M_i\ res_i)$
5 $(\lambda(lb_f,ub_f).\ \exists lb\ ub.\ A_{lb}^{out}\ lb\ lb_f \star A_{ub}^{out}\ ub\ ub_f$
6 $\star \uparrow(\forall s \in states\ M.\ lb\ s \leq P_{\text{inf}}\ M\ s \leq ub\ s))$

Lines 2 and 3 specify the preconditions, where Line 2 states the input data: n and L are natural numbers implemented as 64-bit words n_i and L_i; ε is a real number implemented as float ε_f and M is the MDP implemented as M_i. The separation conjunction $\star$ specifies that these implementations do not overlap on the heap. Line 3 is a boolean predicate lifted to separation logic using $\uparrow$. It states that M satisfies locale *MDP_Reach* (Locale 3.1) and limits the number of states to the largest 64-bit number.

If these preconditions hold, a run of the algorithm (Line 4) yields the arrays lb_f and ub_f that satisfy the postconditions in Lines 5 and 6. These state that lb_f is a lower-bound implementation of lb, which is in turn a lower bound of P_{inf} (similarly for ub). Due to this last fact induced by the refinement, we cannot guarantee convergence of lb_f and ub_f. However, we experimentally show that convergence is generally achieved with our implementation.

5 Experimental Evaluation

The LLVM code generator of the IRF exports the LLVM program *ii_gs_inf_llvm* for use in the LLVM compiler pipeline. Additionally, it generates a C header that makes it easy to embed the program into other software, such as mcsta.

Integration with mcsta. We integrate our verified implementation into mcsta, replacing the existing unverified interval iteration algorithm. This requires a small amount of unverified glue code to convert the MDP's probabilities into a floating-point representation: mcsta stores the probabilities in the MDP as 128-bit rationals (encoded as a pair of 64-bit integers representing the numerator and denominator). Our MDP data structure M_i expects two floats per branch, representing lower and upper bounds of the rational probability. We convert the probabilities to 64-bit doubles by first converting the numerator and denominator to doubles and then performing two separate division operations, once rounding up and and once down. We assume that the input MDP as produced by the mcsta pipeline is well-formed. If there were a bug in the parser that produces MDPs that are not well-formed according to the precondition of the Hoare triple, we would lose the formal guarantees. As long as parts of the toolchain remain unverified, we rely on the correctness of the mcsta implementation for the preprocessing.

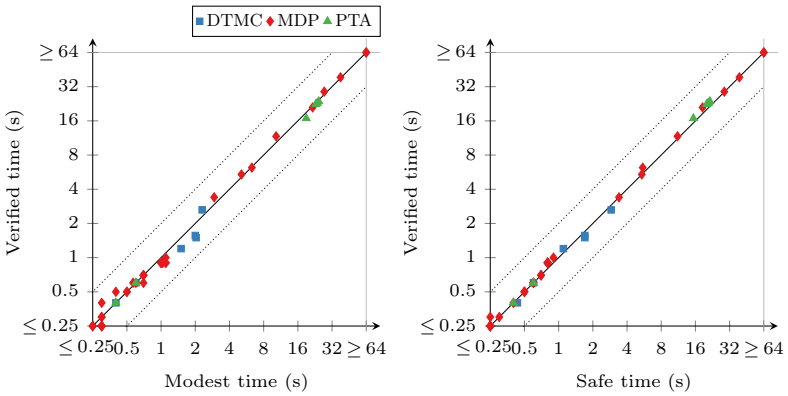


Fig. 5. Comparison of elapsed runtime for completing Interval Iteration

Evaluation Questions. We have formally verified in Isabelle/HOL that II with precise arithmetic computes lower and upper bounds (lb and ub), and eventually converges to the reachability probability. However, two important questions remain: First, verified implementations tend to be orders of magnitude slower than unverified ones [50]. Since we verified an implementation with efficient numerics down to LLVM, we expect much faster runtimes. So how does our verified implementation compare to state-of-the-art tools in terms of performance? Second, the floating-point outputs (lb_f and ub_f) provably provide conservative bounds. Can we experimentally confirm that the algorithm converges in practice?

Setup. For the first question, we compare the runtime of our verified II implementation to its two unverified counterparts in mcsta: a C# implementation with standard rounding (*Modest* implementation) and a C implementation with safe rounding [26] (*Safe* implementation). The latter is similar to our verified LLVM implementation, also using AVX512 instructions for safe rounding.

We set the maximal iteration count to a high value (10^7) to ensure the computation is never terminated prematurely before convergence. While we anticipate all benchmarks to converge within fewer iterations, this upper limit provides a safeguard. We set the convergence threshold to $\varepsilon = 10^{-6}$. Once the lower and upper bound differ by less than ε, the *Verified*, *Safe*, and *Modest* implementations terminate. We use all DTMC, MDP and PTA models of the Quantitative Verification Benchmark Set (QVBS) [15] that contain 10^6 to 10^8 states and need at least two iterations to converge to ε. For our benchmarks, we consider both minimal and maximal reachability probabilities. In total, this yields a benchmark set of 49 benchmark instances. We execute all benchmarks on an Intel i9-11900K (3.5-5.3 GHz) system with 128 GB of RAM running Ubuntu Linux 22.04.

Results. Fig. 5 compares the runtimes of the three implementations. Each point $\langle x, y \rangle$ in a plot indicates that, on one benchmark instance, the implementation on the horizontal axis took x seconds while the *Verified* implementation took y

seconds. We note that none of our benchmark instances reached the timeout of 10 minutes. The times are for running the II algorithm only and do not include other steps mcsta performs equally for all implementations such as state space exploration or min/max-reduction.

We see that the *Verified* implementation matches the unverified ones in terms of performance. Thus, as far as this benchmark set can show, the answer to the first question is that we have achieved comparable performance to a state-of-the-art unverified tool with a fully verified implementation. We also observed that the *Verified* implementation does not reach the maximal iteration count on any instance, i.e. it always converged up to ε, indicating that the second question can also be answered affirmatively.

Additionally, we did not find any significant discrepancies in the raw data output. The number of iterations to convergence is equal for all instances except for the Haddad-Monmege model [22] between *Verified* and *Modest*. The exception is not surprising, as this model is designed to converge very slowly, increasing the influence of floating-point errors. We also compared the computed results. Due to using different rounding modes, the results of *Verified* and *Modest* show differences well within ε. Despite the fact that *Verified* and *Safe* both implement safe rounding, their outputs still differ by minimal amounts on the order of 10^{-20}. This may be caused by floating-point operations being non-associative, so a different order of operations may yield different outputs.

In terms of memory consumption, *Verified* and *Safe* use almost the same amount of RAM, but use on average 20% more RAM than *Modest*. Since the memory consumption is very similar for *Verified* and *Safe*, we suspect that these differences come from garbage collection effects caused by *Verified* and *Safe* being native code called from within a tool otherwise running in the managed C# runtime, as opposed to the purely-C# *Modest* implementation.

6 Discussion

We have formally verified the interval iteration algorithm in Isabelle/HOL. Our developments prove that the algorithm computes lower and upper bounds for the reachability probabilities (soundness) and converges to a single fixpoint (completeness). Furthermore, we show that soundness is preserved if we implement the algorithm using floating-point arithmetic with safe rounding. For this purpose, we used a principled refinement approach. We exploited the parametricity principle [56] of the IRF by consistently rounding our floating-point values in one direction. To make this practical, we equipped the sepref tool with reasoning infrastructure for floating-point numbers to generate an LLVM program. All our proofs culminate in a single statement, presented as a Hoare triple, leaving no gaps in the link between the specification and the implementation in LLVM.

Finally, we extract verified LLVM code from our formalization and embed it in the mcsta model checker of the Modest toolset. We experimentally verify that our implementation converges in practice and is competitive with manually im-

plemented, optimised unverified counterparts. This is an important step towards a fully verified probabilistic model checking pipeline.

We also present our approach as an alternative to the bottom-up approach of building a verified model checker from scratch. With our top-down approach, the full functionality of the model checker is available to the user, possibly in cross-usage with verified components. Verified components are integrated with the model checker incrementally as drop-in replacements for unverified components, designed with competitive performance in mind.

Verification vs. Certification for II. An alternative to verifying II is to verify a *certifier* that, given the result from an unverified implementation of II plus a (compact) *certificate*, can efficiently check that the result is indeed correct. The advantage of certification is that the unverified implementation of II can be improved in an agile way independent of the certifier. Also since the certifier is (presumably) simpler than II, it should be easier to verify, including the verification of a high-performance implementation. One possible certification scheme for II is using the Park induction check of the optimistic value iteration algorithm [29], i.e. performing one iteration of II and checking if the lower/upper bound does not decrease/increase for any state. This could confirm that *some* fixed point lies between the bounds computed by II. This certification scheme would require as input (1) the final lower and upper bound for all states, (2) the full MDP, (3) *and a certificate that the MDP is reduced* that it also needs to check. Without the latter, one could not be sure that the certified fixed point corresponds to the reachability probability we want to compute (i.e. that it is the least fixed point). One challenge with this scheme is that, unlike our verified implementation of II, it cannot guarantee that *all* fixed points lie between the bounds. Also, since the entire reduced MDP is input to the certificate checker, the certificate checking performance might be fundamentally limited.

References

1. Abbasi, R., Schiffl, J., Darulova, E., Ulbrich, M., Ahrendt, W.: Deductive verification of floating-point java programs in KeY. In: Tools and Algorithms for the Construction and Analysis of Systems - 27th International Conference, TACAS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 - April 1, 2021, Proceedings, Part II. pp. 242–261 (2021). https://doi.org/10.1007/978-3-030-72013-1_13
2. Ahrendt, W., Beckert, B., Bubel, R., Hähnle, R., Schmitt, P.H., Ulbrich, M. (eds.): Deductive Software Verification - the KeY Book - from Theory to Practice (2016)
3. Baier, C.: Probabilistic model checking. In: Dependable Software Systems Engineering, pp. 1–23. IOS Press (2016). https://doi.org/10.3233/978-1-61499-627-9-1
4. Baier, C., de Alfaro, L., Forejt, V., Kwiatkowska, M.: Model checking probabilistic systems. In: Handbook of Model Checking, pp. 963–999. Springer (2018). https://doi.org/10.1007/978-3-319-10575-8_28

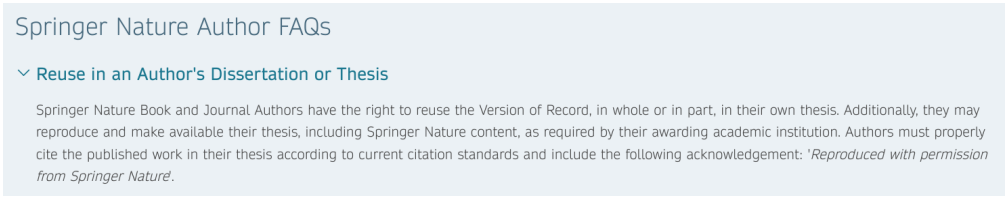
5. Baier, C., Haverkort, B.R., Hermanns, H., Katoen, J.P.: Performance evaluation and model checking join forces. Communications of The Acm (9), 76–85 (2010). https://doi.org/10.1145/1810891.1810912
6. Baier, C., Katoen, J.P.: Principles of Model Checking. MIT Press (2008)
7. Baier, C., Klein, J., Leuschner, L., Parker, D., Wunderlich, S.: Ensuring the reliability of your model checker: Interval iteration for Markov decision processes. In: 29th International Conference on Computer Aided Verification (CAV). pp. 160–180 (2017). https://doi.org/10.1007/978-3-319-63387-9_8
8. Ballarin, C.: Locales and locale expressions in Isabelle/Isar. In: TYPES. pp. 34–50 (2003)
9. Barrett, G.: Formal methods applied to a floating-point number system. IEEE Transactions on Software Engineering (5), 611–621 (1989). https://doi.org/10.1109/32.24710
10. Bellman, R.: A Markovian decision process. Journal of Mathematics and Mechanics (5), 679–684 (1957)
11. Bohnenkamp, H.C., D'Argenio, P.R., Hermanns, H., Katoen, J.P.: MoDeST: A compositional modeling formalism for hard and softly timed systems. IEEE Transactions on Software Engineering (10), 812–830 (2006). https://doi.org/10.1109/TSE.2006.104
12. Boldo, S., Melquiond, G.: Flocq: A Unified Library for Proving Floating-Point Algorithms in Coq. In: 2011 IEEE 20th Symposium on Computer Arithmetic. pp. 243–252 (2011). https://doi.org/10.1109/ARITH.2011.40
13. Boldo, S., Munoz, C.: A high-level formalization of floating-point number in PVS. Tech. rep. (2006)
14. Budde, C.E., Dehnert, C., Hahn, E.M., Hartmanns, A., Junges, S., Turrini, A.: JANI: Quantitative model and tool interaction. In: 23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). pp. 151–168 (2017). https://doi.org/10.1007/978-3-662-54580-5_9
15. Budde, C.E., Hartmanns, A., Klauck, M., Kretínský, J., Parker, D., Quatmann, T., Turrini, A., Zhang, Z.: On correctness, precision, and performance in quantitative verification (QComp 2020 competition report). In: 9th International Symposium on Leveraging Applications of Formal Methods (ISoLA). pp. 216–241 (2020). https://doi.org/10.1007/978-3-030-83723-5_15
16. Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., Monniaux, D., Rival, X.: The ASTREÉ analyzer. In: Programming Languages and Systems, 14th European Symposium on Programming, ESOP 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005, Proceedings. pp. 21–30 (2005). https://doi.org/10.1007/978-3-540-31987-0_3
17. D'Argenio, P.R., Jeannet, B., Jensen, H.E., Larsen, K.G.: Reachability analysis of probabilistic systems by successive refinements. In: Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification (PAPM-PROBMIV). pp. 39–56 (2001). https://doi.org/10.1007/3-540-44804-7_3
18. de Alfaro, L.: Formal Verification of Probabilistic Systems. Ph.D. thesis, Stanford University, USA (1997)
19. De Dinechin, F., Lauter, C., Melquiond, G.: Certifying the floating-point implementation of an elementary function using Gappa. IEEE Transactions on Computers (2), 242–253 (2010). https://doi.org/10.1109/TC.2010.128

20. Goubault, E., Putot, S.: Static analysis of finite precision computations. In: Verification, Model Checking, and Abstract Interpretation - 12th International Conference, VMCAI 2011, Austin, TX, USA, January 23-25, 2011. Proceedings. pp. 232–247 (2011). https://doi.org/10.1007/978-3-642-18275-4_17
21. Haddad, S., Monmege, B.: Reachability in MDPs: Refining convergence of value iteration. In: 8th International Workshop on Reachability Problems (RP). pp. 125–137 (2014). https://doi.org/10.1007/978-3-319-11439-2_10
22. Haddad, S., Monmege, B.: Interval iteration algorithm for mdps and imdps. Theoretical Computer Science pp. 111–131 (2018). https://doi.org/10.1016/J.TCS.2016.12.003
23. Hahn, E.M., Hartmanns, A., Hermanns, H., Katoen, J.P.: A compositional modelling and analysis framework for stochastic hybrid systems. Formal Methods Syst. Des. (2), 191–232 (2013). https://doi.org/10.1007/S10703-012-0167-Z
24. Harrison, J.: Formal verification at Intel. In: 18th Annual IEEE Symposium of Logic in Computer Science, 2003. pp. 45–54 (2003). https://doi.org/10.1109/LICS.2003.1210044
25. Harrison, J.: A Machine-Checked Theory of Floating Point Arithmetic. In: Theorem Proving in Higher Order Logics. pp. 113–130 (1999). https://doi.org/10.1007/3-540-48256-3_9
26. Hartmanns, A.: Correct probabilistic model checking with floating-point arithmetic. In: TACAS (2). pp. 41–59 (2022). https://doi.org/10.1007/978-3-030-99527-0_3
27. Hartmanns, A., Hermanns, H.: The Modest Toolset: An integrated environment for quantitative modelling and verification. In: 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). pp. 593–598 (2014). https://doi.org/10.1007/978-3-642-54862-8_51
28. Hartmanns, A., Junges, S., Quatmann, T., Weininger, M.: A practitioner's guide to MDP model checking algorithms. In: 29th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). pp. 469–488 (2023). https://doi.org/10.1007/978-3-031-30823-9_24
29. Hartmanns, A., Kaminski, B.L.: Optimistic value iteration. In: 32nd International Conference on Computer Aided Verification (CAV). pp. 488–511 (2020). https://doi.org/10.1007/978-3-030-53291-8_26
30. Hartmanns, A., Klauck, M., Parker, D., Quatmann, T., Ruijters, E.: The quantitative verification benchmark set. In: 25th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). pp. 344–350 (2019). https://doi.org/10.1007/978-3-030-17462-0_20
31. Hartmanns, A., Kohlen, B., Lammich, P.: Fast verified sccs for probabilistic model checking. In: ATVA (1). pp. 181–202 (2023). https://doi.org/10.1007/978-3-031-45329-8_9
32. Hartmanns, A., Kohlen, B., Lammich, P.: Efficient formally verified maximal end component decomposition for mdps. In: FM (1). pp. 206–225 (2024). https://doi.org/10.1007/978-981-99-8664-4
33. Hölzl, J.: Markov chains and markov decision processes in isabelle/HOL. Journal of Automated Reasoning (3), 345–387 (2017). https://doi.org/10.1007/S10817-016-9401-5
34. Hölzl, J., Nipkow, T.: Markov models. Arch. Formal Proofs (2012)
35. Kamali, M., Katoen, J.P.: Probabilistic model checking of AODV. In: 17th International Conference on the Quantitative Evaluation of Systems (QEST). pp. 54–73 (2020). https://doi.org/10.1007/978-3-030-59854-9_6

36. Kirchner, F., Kosmatov, N., Prevosto, V., Signoles, J., Yakobowski, B.: Frama-C: A software analysis perspective. Formal Aspects Comput. (3), 573–609 (2015). https://doi.org/10.1007/S00165-014-0326-7
37. Kosmatov, N., Prevosto, V., Signoles, J.: Guide to Software Verification with Frama-c. Springer (2024)
38. Lammich, P.: Generating verified LLVM from isabelle/HOL. In: 10th International Conference on Interactive Theorem Proving (ITP). pp. 22:1–22:19 (2019). https://doi.org/10.4230/LIPICS.ITP.2019.22
39. Lammich, P., Tuerk, T.: Applying data refinement for monadic programs to hopcroft's algorithm. In: 3rd International Conference on Interactive Theorem Proving (ITP). pp. 166–182 (2012). https://doi.org/10.1007/978-3-642-32347-8_12
40. Lattner, C., Adve, V.: LLVM: A compilation framework for lifelong program analysis and transformation. In: CGO. pp. 75–88 (2004)
41. Magron, V., Constantinides, G.A., Donaldson, A.F.: Certified roundoff error bounds using semidefinite programming. ACM Trans. Math. Softw. (4), 34:1–34:31 (2017). https://doi.org/10.1145/3015465
42. Miner, P.S.: Defining the IEEE-854 floating-point standard in PVS. Tech. rep. (1995)
43. Moscato, M., Titolo, L., Dutle, A., Muñoz, C.A.: Automatic Estimation of Verified Floating-Point Round-Off Errors via Static Analysis. In: Computer Safety, Reliability, and Security. pp. 213–229 (2017). https://doi.org/10.1007/978-3-319-66266-4_14
44. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL – a Proof Assistant for Higher-Order Logic. Springer (2002)
45. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley (1994). https://doi.org/10.1002/9780470316887
46. Quatmann, T., Katoen, J.P.: Sound value iteration. In: 30th International Conference on Computer Aided Verification (CAV). pp. 643–661 (2018). https://doi.org/10.1007/978-3-319-96145-3_37
47. Roberts, R., Lewis, B., Hartmanns, A., Basu, P., Roy, S., Chakraborty, K., Zhang, Z.: Probabilistic verification for reliability of a two-by-two network-on-chip system. In: 26th International Conference on Formal Methods for Industrial Critical Systems (FMICS). pp. 232–248 (2021). https://doi.org/10.1007/978-3-030-85248-1_16
48. Ruijters, E., Guck, D., Drolenga, P., Peters, M., Stoelinga, M.: Maintenance analysis and optimization via statistical model checking – evaluating a train pneumatic compressor. In: 13th International Conference on the Quantitative Evaluation of Systems (QEST). pp. 331–347 (2016). https://doi.org/10.1007/978-3-319-43425-4_22
49. Ruijters, E., Guck, D., van Noort, M., Stoelinga, M.: Reliability-centered maintenance of the electrically insulated railway joint via fault tree analysis: A practical experience report. In: 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). pp. 662–669 (2016). https://doi.org/10.1109/DSN.2016.67
50. Schäffeler, M., Abdulaziz, M.: Formally Verified Solution Methods for Infinite-Horizon Markov Decision Processes. In: The 37th AAAI Conference on Artificial Intelligence (AAAI) (2023). https://doi.org/10.1609/aaai.v37i12.26759
51. Schäffeler, M., Abdulaziz, M.: Formally Verified Approximate Policy Iteration. In: The 38th AAAI Conference on Artificial Intelligence (AAAI), To Appear (2025). https://doi.org/10.48550/arXiv.2406.07340

52. Solovyev, A., Baranowski, M.S., Briggs, I., Jacobsen, C., Rakamaric, Z., Gopalakrishnan, G.: Rigorous estimation of floating-point round-off errors with symbolic taylor expansions. ACM Trans. Program. Lang. Syst. (1), 2:1–2:39 (2019). https://doi.org/10.1145/3230733
53. Sutton, R.S., Barto, A.G.: Reinforcement Learning – an Introduction. MIT Press (1998)
54. Titolo, L., Moscato, M., Feliu, M.A., Masci, P., Muñoz, C.A.: Rigorous Floating-Point Round-Off Error Analysis in PRECiSA 4.0. In: Formal Methods. pp. 20–38 (2025). https://doi.org/10.1007/978-3-031-71177-0_2
55. Vajjha, K., Shinnar, A., Trager, B.M., Pestun, V., Fulton, N.: CertRL: Formalizing convergence proofs for value and policy iteration in Coq. In: The 10th International Conference on Certified Programs and Proofs (CPP). pp. 18–31 (2021). https://doi.org/10.1145/3437992.3439927
56. Wadler, P.: Theorems for Free! In: Proceedings of the Fourth International Conference on Functional Programming Languages and Computer Architecture, FPCA 1989, London, UK, September 11-13, 1989. pp. 347–359 (1989). https://doi.org/10.1145/99370.99404
57. Wimmer, R., Kortus, A., Herbstritt, M., Becker, B.: Probabilistic model checking and reliability of results. In: 11th IEEE Workshop on Design & Diagnostics of Electronic Circuits & Systems (DDECS). pp. 207–212 (2008). https://doi.org/10.1109/DDECS.2008.4538787
58. Yu, L.: A formal model of IEEE floating point arithmetic. Arch. Formal Proofs (2013)

Copyright and License Information


Springer Nature Author FAQs

Reuse in an Author's Dissertation or Thesis

Springer Nature Book and Journal Authors have the right to reuse the Version of Record, in whole or in part, in their own thesis. Additionally, they may reproduce and make available their thesis, including Springer Nature content, as required by their awarding academic institution. Authors must properly cite the published work in their thesis according to current citation standards and include the following acknowledgement: '*Reproduced with permission from Springer Nature*'.

Figure 1: Springer Nature permission, retrieved from `https://www.springernature.com/gp/partners/rights-permissions-third-party-distribution` (accessed on April 10, 2025)

Core Publication 4

Krishnendu Chatterjee, Tim Quatmann, Maximilian Schäffeler, Maximilian Weininger, Tobias Winkler, and Daniel Zilken. "Fixed Point Certificates for Reachability and Expected Rewards in MDPs." In: *Tools and Algorithms for the Construction and Analysis of Systems*. Lecture Notes in Computer Science. Springer, 2025. To appear.

Fixed Point Certificates for Reachability and Expected Rewards in MDPs*

Krishnendu Chatterjee[1], Tim Quatmann[2], Maximilian Schäffeler[3], Maximilian Weininger[1], Tobias Winkler[2], and Daniel Zilken[1,2]

[1] Institute of Science and Technology Austria, Klosterneuburg, Austria · {Krishnendu.Chatterjee, Maximilian.Weininger}@ist.ac.at
[2] RWTH Aachen, Germany · {tim.quatmann, tobias.winkler, daniel.zilken}@cs.rwth-aachen.de
[3] Technical University of Munich, Germany · maximilian.schaeffeler@tum.de

Abstract. The possibility of errors in human-engineered formal verification software, such as model checkers, poses a serious threat to the purpose of these tools. An established approach to mitigate this problem are *certificates*—lightweight, easy-to-check proofs of the verification results. In this paper, we develop novel certificates for model checking of *Markov decision processes* (MDPs) with quantitative reachability and expected reward properties. Our approach is conceptually simple and relies almost exclusively on elementary fixed point theory. Our certificates work for *arbitrary* finite MDPs and can be readily computed with little overhead using standard algorithms. We formalize the soundness of our certificates in Isabelle/HOL and provide a *formally verified certificate checker*. Moreover, we augment existing algorithms in the probabilistic model checker Storm with the ability to produce certificates and demonstrate practical applicability by conducting the first formal certification of the reference results in the Quantitative Verification Benchmark Set.

Keywords: Probabilistic model checking · Markov decision processes · Certificates · Reachability · Expected rewards · Proof assistant

1 Introduction

Markov decision processes (MDPs) [48,7,5] are *the* model for sequential decision making in probabilistic environments. Their many applications [53,32] frequently require computing *reachability probabilities* towards an (un-)desired system state, as well as the *expected rewards* (or costs) accumulated until doing so. *MDP model checking* amounts to computing (approximations of) these

* This project has received funding from the ERC CoG 863818 (ForM-SMArt), the Austrian Science Fund (FWF) 10.55776/COE12, a KI-Starter grant from the Ministerium für Kultur und Wissenschaft NRW, the DFG RTG 378803395 (ConVeY), the EU's Horizon 2020 research and innovation programmes under the Marie Sklodowska-Curie grant agreement Nos. 101034413 (IST-BRIDGE) and 101008233 (MISSION), and the DFG RTG 2236 (UnRAVeL). Experiments were performed with computing resources granted by RWTH Aachen University under project rwth1632.

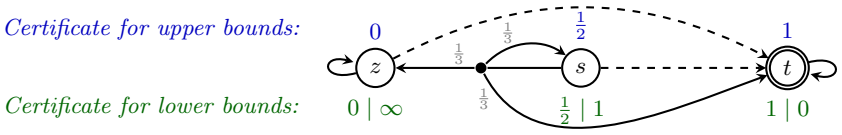


Fig. 1. An MDP with states $S = \{z, s, t\}$, two actions (distinguished by solid and dashed edges), uniform probabilities, and target set $T = \{t\}$. The annotations above and below each state are a certificate for upper and lower bounds on $\mathbb{P}^{\min}(\Diamond T)$, resp.

quantities in a *mathematically rigorous* way, with a formal guarantee of their correctness and precision. Various mature MDP model checking tools such as PRISM [42], mcsta [28], and Storm [33] exist. Figure 1 shows an example MDP.

Who checks the model checker? The possibility of errors in complex, human-engineered formal verification tools is a delicate issue: How *formal* is a verification result produced by an *informal*, i.e., unverified implementation? We highlight four sources of errors: (i) classic implementation bugs, (ii) unintentionally unsound algorithms [8,24], optimizations, and heuristics, (iii) numerical errors due to floating point arithmetic [27], and (iv) errors in third-party back end libraries or tools, e.g., commercial LP solvers [29].

Certifying algorithms [44] are a paradigm for establishing trust in implementations. A certifying algorithm produces a concise, easily verifiable proof—a *certificate*—of its result. The certificate can be checked independently, possibly even by an external, simpler program amenable to formal verification, or by a third party. Formally verified certificate checkers are already employed in tool competitions on software verification [12] or SAT-solving [9]. Existing proposals for certifying MDPs [34,22,35], however, have some drawbacks (detailed further below) hindering wider adoption in the community and its competitions [26,15,3].

The goal of this paper is to establish a new standard for certified MDP model checking, with a focus on applicability and extensibility.

Our contributions towards this goal are as follows:

- We present *fixed point certificates* for two-sided bounds on extremal reachability probabilities (Table 1) and expected rewards. Our certificates are sound and complete for *arbitrary* finite MDPs without structural restrictions.
- We formalize the theory in Isabelle/HOL [46], proving soundness of our certificates, and generate a *formally verified certificate checker implementation*.
- We implement a certifying variant of [29] *Interval Iteration* [8] with floating point arithmetic in Storm [33]. Using this, we give certified reference results for the Quantitative Verification Benchmark Set [32].

Extensibility towards further properties is enabled by our simple, clean theory summarized as four *guiding principles*: (GP1) We characterize the quantities of interest as a *fixed point* of basic, easy-to-evaluate *Bellman-type* operator [11]. The fundamental certification mechanism is to use *fixed point induction* for proving

Table 1. Our reachability certificates. Sound and complete for arbitrary finite MDPs.

Certificate	Condition(s)	*Explanation*
Upper bounds on **minimal** reachability probabilities: $\forall s \in S\colon \mathbb{P}_s^{\min}(\Diamond T) \leq x(s)$ [Proposition 3]		
$x \in [0,1]^S$	$\mathcal{B}^{\min}(x) \leq x$	***min**-Bellman operator **decreases** value of all states*
Upper bounds on **maximal** reachability probabilities: $\forall s \in S\colon \mathbb{P}_s^{\max}(\Diamond T) \leq x(s)$ [Proposition 3]		
$x \in [0,1]^S$	$\mathcal{B}^{\max}(x) \leq x$	***max**-Bellman operator **decreases** value of all states*
Lower bounds on **minimal** reachability probabilities: $\forall s \in S\colon \mathbb{P}_s^{\min}(\Diamond T) \geq x(s)$ [Proposition 4]		
$x \in [0,1]^S$	$\mathcal{B}^{\min}(x) \geq x$	***min**-Bellman operator **increases** value of all states*
$r \in \overline{\mathbb{N}}^S$	$\mathcal{D}^{\max}(r) \leq r$	*r upper bounds **maximal** distances to T*
	$x(s){>}0 \implies r(s){<}\infty$	*positive reachability necessitates finite distance*
Lower bounds on **maximal** reachability probabilities: $\forall s \in S\colon \mathbb{P}_s^{\max}(\Diamond T) \geq x(s)$ [Proposition 6]		
$x \in [0,1]^S$	$\mathcal{B}^{\max}(x) \geq x$	***max**-Bellman operator **increases** value of all states*
$r \in \overline{\mathbb{N}}^S$	$\mathcal{D}^{\min}_{x\uparrow}(r) \leq r$	*r upper bounds **min.** distances to T via x-**incr. actions***
	$x(s){>}0 \implies r(s){<}\infty$	*positive reachability necessitates finite distance*

bounds on the least or greatest fixed point. (GP2) We certify *qualitative* reachability properties using *ranking functions* which are amenable to fixed point induction, too. (GP3) As the basic Bellman operators frequently have undesired, spurious fixed points [13,24,39] (often related to *end components* [2]), we consider slight modifications requiring qualitative reachability information which we certify following GP2. (GP4) When GP3 is insufficient or not applicable, we implicitly include a *witness strategy* in our certificate.

Technical challenges still arise in concretely applying these guiding principles. For instance, a key novelty of our paper is a ranking-function type certificate for *not almost sure reachability* (Proposition 2), which is surprisingly involved.

Related work. Closest to our work are the previous proposals for certificates in MDP model checking: [34, Sec. 4] formally verifies a theory of certificates for reachability objectives, which is however limited to upper bounds on maximal and lower bounds on the minimal probabilities. [22] presents so-called "Farkas certificates" for reachability; however, it does not offer a formally verified certificate checker, is limited to MDPs without end components (ECs), and does not address certificate generation explicitly. With similar limitations, [6] provides Farkas certificates for *multiple* reachability or mean payoff objectives, which can be computed via linear programming. [35] suggests lifting the EC assumptions from [22] by certifying the full maximal EC decomposition. In contrast, our certificates are more concise as they handle ECs using at most one ranking function. Further, [35] proposes certificates for expected rewards, but they require the target to be reached almost surely, an assumption we do not have to make.

Witnessing subsystems [54,22,36,6] are an alternative certification paradigm. However, their verification requires more computational effort than the simple, state-wise operations needed for checking Farkas or fixed point certificates. Still, they utilize similar ideas: The backward reachable states in [54, Sec. 3.3.3] essentially use ranking functions, as do the constraints in [37, Sec. 5.2.2].

The term "certifying algorithms" was coined in [38]. Previous work on certificates for other verification problems includes [45,47,41,20,40]. Further, certificates were recently investigated in hardware verification [57,58,59,21] and

approximate model counting [52]. Finally, we mention that *Optimistic Value Iteration* (OVI) [31,4], the supermartingales in [43,1,51], the certificates for probabilistic pushdown systems from [55,56], and a recent strategy synthesis method for infinite MDPs [10] follow fixed point induction principles similar to GP1.

Paper outline. After the background on MDPs and fixed point theory (Section 2), we introduce ranking functions for qualitative reachability (Section 3). Building on this, we discuss quantitative reachability (Section 4, Table 1) and expected rewards (Section 5, [17, Tab. 2]). We explain how to compute certificates (Section 6) and report on experiments (Section 7). Omitted pen-and-paper proofs are in [17, App. C–E]. *All proofs regarding the soundness of the certificates, even standard results from the literature, are formalized in* Isabelle/HOL.

2 Preliminaries

A *Markov decision processes* (MDP) is a tuple $\mathcal{M} = (S, Act, P)$ where S is a finite set of *states*, Act is a finite set of *actions*, and $P\colon S \times Act \times S \to [0,1]$ is a *transition probability function* with the property that $\sum_{s' \in S} P(s,a,s') \in \{0,1\}$ for all $s \in S$ and $a \in Act$. For every $s \in S$, the set of *enabled* actions $a \in Act$ for which the above sum equals 1 is written $Act(s)$. It is required that $Act(s) \neq \emptyset$ for all $s \in S$. For $s \in S$ and $a \in Act(s)$, we define the *a-successors* of s as $Post(s,a) = \{s' \in S \mid P(s,a,s') > 0\}$. Notice that our MDPs do not have a distinguished initial state. See Figure 1 for an example MDP.

A (finite-state, discrete-time) *Markov chain* (DTMC) is the special case of an MDP with $|Act(s)| = 1$ for all $s \in S$. A (memoryless and deterministic) *strategy*[4] for an MDP $\mathcal{M} = (S, Act, P)$ is a function $\sigma\colon S \to Act$ such that for all $s \in S$ we have $\sigma(s) \in Act(s)$. We may apply σ to $\mathcal{M}$ to obtain the *induced DTMC* $\mathcal{M}^\sigma = (S, Act, P^\sigma)$ which, intuitively, only retains the actions chosen by σ. Formally, for all $s, s' \in S$ we define $P^\sigma(s, \sigma(s), s') = P(s, \sigma(s), s')$, and $P^\sigma(s,a,s') = 0$ for all $a \neq \sigma(s)$.

Reachability and Expected Rewards. Fix a DTMC (S, Act, P), a *target* set $T \subseteq S$, and a *reward function* $\mathrm{rew}\colon S \to \mathbb{R}_{\geq 0}$. We define two random variables $\Diamond T$ and $\mathrm{rew}^{\Diamond T}$ taking values in $\{0,1\}$ and $\overline{\mathbb{R}}_{\geq 0}$, respectively: For $s_0 s_1 ... \in S^\omega$ an infinite path, we set $\Diamond T(s_0 s_1 \ldots) = 1$ if and only if (iff) $\exists i \in \mathbb{N}\colon s_i \in T$. Moreover:

$$\mathrm{rew}^{\Diamond T}(s_0 s_1 \ldots) \;=\; \begin{cases} \sum_{k=0}^{\min\{i \mid s_i \in T\}} \mathrm{rew}(s_k) & \text{if } \exists i \in \mathbb{N}\colon s_i \in T \\ * & \text{else} \end{cases}$$

We consider both options $* = \infty$ and $* = \sum_{k=0}^{\infty} \mathrm{rew}(s_k)$ [19]. We focus on the former in the main body, as it is standard in the literature [7, Def. 10.71] and tool competitions [26,15]; we treat the latter in [17, App. F]. Intuitively, with $* = \infty$, $\mathrm{rew}^{\Diamond T}$ assigns ∞ to paths that never reach T. The other paths receive the sum of rewards collected until reaching T for the first time. Given a state $s \in S$,

[4] Aka. scheduler or policy. We do not define more general strategies as memoryless deterministic suffice for optimizing reachability probabilities and expected rewards.

we define the *reachability probability* $\mathbb{P}_s(\Diamond T)$ from s to T as the expected value (Lebesgue integral) of $\Diamond T$ w.r.t. the probability measure $\mathbb{P}_s$ on infinite paths of the DTMC with initial state fixed to s, see [7, Ch. 10] for the construction of $\mathbb{P}_s$. Similarly, the *expected reward* $\mathbb{E}_s(\mathrm{rew}^{\Diamond T})$ from s to T is the expected value of $\mathrm{rew}^{\Diamond T}$. When rew is clear from context, we write $\mathbb{E}_s(\Diamond T)$ instead of $\mathbb{E}_s(\mathrm{rew}^{\Diamond T})$.

Finally, given an MDP $\mathcal{M} = (S, Act, P)$, a state $s \in S$, a target set $T \subseteq S$, a reward function $\mathrm{rew}\colon S \to \mathbb{R}_{\geq 0}$, and $\mathrm{opt} \in \{\min, \max\}$ we define the *optimal reachability probability* $\mathbb{P}_s^{\mathrm{opt}}(\Diamond T) = \mathrm{opt}_\sigma \mathbb{P}_s^\sigma(\Diamond T)$ and the *optimal expected reward* $\mathbb{E}_s^{\mathrm{opt}}(\mathrm{rew}^{\Diamond T}) = \mathrm{opt}_\sigma \mathbb{E}_s^\sigma(\mathrm{rew}^{\Diamond T})$, where $\mathbb{P}_s^\sigma(\Diamond T)$ and $\mathbb{E}_s^\sigma(\mathrm{rew}^{\Diamond T})$ are the reachability probabilities and expected rewards in the induced DTMC $\mathcal{M}^\sigma$.

Fixed Point Theory. A *partial order* on a set X is a binary relation $\preceq$ that is reflexive, transitive, and antisymmetric; in this case, the tuple $(X, \preceq)$ is called a *poset*. Given a poset $(X, \preceq)$, we call $a \in X$ an *upper bound* on $Y \subseteq X$ if for all $b \in Y$ we have $b \preceq a$. If an upper bound a on Y is minimal among all upper bounds, it is the unique *supremum* (or least upper bound) and denoted $\sup Y$. *Lower bounds* and *infima* (or greatest lower bounds) are defined analogously.

The poset $(X, \preceq)$ is a *complete lattice* if $\sup Y$ and $\inf Y$ exist for every $Y \subseteq X$. Every complete lattice has a least and greatest element $\sup \emptyset$ and $\inf \emptyset$, respectively. The following complete lattices are of interest in this paper:

- $(\overline{\mathbb{N}}, \leq)$ where $\overline{\mathbb{N}} = \mathbb{N} \cup \{\infty\}$ are the *extended natural numbers* and $\leq$ is the usual order on $\mathbb{N}$ extended by $a \leq \infty$ for all $a \in \overline{\mathbb{N}}$. Notice that for every $Y \subseteq \mathbb{N}$, $\sup Y = \infty$ iff Y is infinite.
- Similarly, $(\overline{\mathbb{R}}_{\geq 0}, \leq)$, with $\overline{\mathbb{R}}_{\geq 0} = \mathbb{R}_{\geq 0} \cup \{\infty\}$ the *extended non-negative reals*, is a complete lattice. For every $Y \subseteq \mathbb{R}_{\geq 0}$, $\sup Y = \infty$ iff Y is unbounded.
- $([0,1], \leq)$, the totally ordered set of real probabilities.
- If $(X, \preceq)$ is an arbitrary complete lattice, then for all sets S, $(X^S, \ddot{\preceq})$ is a complete lattice, where $X^S = \{f \mid f\colon S \to X\}$ is the set of functions from S to X and the partial order $\ddot{\preceq}$ is defined as $f \ddot{\preceq} g \iff \forall s \in S\colon f(s) \preceq g(s)$. In the following, we overload notation and write $\preceq$ instead of $\ddot{\preceq}$. For example, if S is the set of states of an MDP, then we can think of $([0,1]^S, \leq)$ as the poset of "probability vectors" indexed by S, partially ordered entry-wise.

Let $(X, \preceq)$ be a poset. A function $\mathcal{F}\colon X \to X$ is called *monotone* if $\forall a, b \in X\colon a \preceq b \implies \mathcal{F}(a) \preceq \mathcal{F}(b)$. The following is *the* key tool of this paper:

Theorem 1 (Knaster-Tarski). *Let $(X, \preceq)$ be a complete lattice and $\mathcal{F}\colon X \to X$ be monotone. Then, the set of* fixed points *$(\{a \in X \mid \mathcal{F}(a) = a\}, \preceq)$ is also a complete lattice. In particular, $\mathcal{F}$ has a* least *and a* greatest *fixed point given by $\mathrm{lfp}\,\mathcal{F} = \inf\{a \in X \mid \mathcal{F}(a) \preceq a\}$ and, dually, $\mathrm{gfp}\,\mathcal{F} = \sup\{a \in X \mid a \preceq \mathcal{F}(a)\}$. As a consequence, the following* fixed point induction rules *are sound: $\forall a \in X$:*

- $\mathcal{F}(a) \preceq a \implies \mathrm{lfp}\,\mathcal{F} \preceq a$ (fixed point induction)
- $a \preceq \mathcal{F}(a) \implies a \preceq \mathrm{gfp}\,\mathcal{F}$ (fixed point co-induction)

Elements $a \in X$ with $\mathcal{F}(a) \preceq a$ (or $a \preceq \mathcal{F}(a)$) are called *(co-)inductive*.

Guiding Principle 1 (Fixed Point Induction) *We apply the theorem of Knaster-Tarski to monotone operators of the form* $\mathcal{F}\colon X^S \to X^S$*, where* X *is a complete lattice and* S *is finite, to certify upper bounds on* $\operatorname{lfp}\mathcal{F}$ *and lower bounds on* $\operatorname{gfp}\mathcal{F}$*. We call such* $\mathcal{F}$ Bellman-type operators.

Throughout the rest of the paper, we fix an MDP $\mathcal{M} = (S, Act, P)$, a set of target states $T \subseteq S$ and, for Section 5, a reward function $\mathrm{rew}\colon S \to \mathbb{R}_{\geq 0}$. Moreover, we let $\mathrm{opt} \in \{\min, \max\}$ and write $\overline{\min} = \max$ and $\overline{\max} = \min$.

3 Certifying Qualitative Reachability

Most of the certificates presented in the forthcoming Sections 4 and 5 enclose a certificate for a *qualitative* reachability property, e.g., $\mathbb{P}^{\mathrm{opt}}(\Diamond T) > 0$ or $\mathbb{P}^{\mathrm{opt}}(\Diamond T) < 1$. Our approach to this is summarized as follows:

Guiding Principle 2 (Ranking Functions) *To certify qualitative reachability properties, we rely on* ranking functions *formalized via appropriate operators capturing certain* distances *in the MDP when viewed as a graph.*

Definition 1 (Distance Operator). *Let* (S, Act, P)*,* T*, and* opt *be the fixed MDP, target set and optimization direction, respectively. (We omit these quantifications in the rest of the paper.) We define the following* distance operator*:*

$$\mathcal{D}^{\mathrm{opt}}\colon\ \overline{\mathbb{N}}^S \to \overline{\mathbb{N}}^S,\ \mathcal{D}^{\mathrm{opt}}(r)(s) = \begin{cases} 0 & \text{if } s \in T \\ 1 \ + \ \underset{a \in Act(s)}{\mathrm{opt}} \ \underset{s' \in Post(s,a)}{\min} \ r(s') & \text{if } s \in S \setminus T \end{cases}$$

$\mathcal{D}^{\mathrm{opt}}$ is a monotone Bellman-type operator on the complete lattice $(\overline{\mathbb{N}}^S, \leq)$ and thus has a least fixed point by Theorem 1. In fact, we even have the following:

Lemma 1 (Unique Fixed Point). $\mathcal{D}^{\mathrm{opt}}$ *has a unique fixed point.*

Intuitively, if $r = \operatorname{fp}\mathcal{D}^{\min}$, then $r(s)$ represents the length of a shortest path from every state $s \in S$ to T, or $r(s) = \infty$ if T is not reachable from s. For $r = \operatorname{fp}\mathcal{D}^{\max}$, $r(s)$ can be seen as the shortest path in the DTMC induced by a strategy that aims to avoid T or reach it as late as possible. We formalize this intuition in Lemma 2 (using the notation $\overline{\min} = \max$ and $\overline{\max} = \min$), and then in Proposition 1 apply Guiding Principle 1 to certify positive reachability.

Lemma 2. *Let* $r = \operatorname{fp}\mathcal{D}^{\mathrm{opt}}$*. Then for all* $s \in S$*,* $r(s) = \infty \iff \mathbb{P}_s^{\overline{\mathrm{opt}}}(\Diamond T) = 0$*.*

Proposition 1 (Certificates for $\mathbb{P}^{\mathrm{opt}}(\Diamond T) > 0$). *A function* $r \in \overline{\mathbb{N}}^S$ *is called a* valid certificate for positive opt-reachability *if* $\mathcal{D}^{\overline{\mathrm{opt}}}(r) \leq r$*. If* r *is valid, then* $\forall s \in S\colon r(s) < \infty \implies \mathbb{P}_s^{\mathrm{opt}}(\Diamond T) > 0$*.*

Example 1. Consider the MDP in Figure 1 on page 2. The values on the bottom right of the states constitute a valid certificate r for positive min-reachability. To check that $\mathcal{D}^{\overline{\min}}(r) = \mathcal{D}^{\max}(r) \leq r$ is indeed true, we verify the following:

$$\mathcal{D}^{\max}(r)(s) \;=\; 1 + \max\Big\{ \underbrace{\min\{0,1,\infty\}}_{\text{solid action}}, \quad \underbrace{0}_{\text{dashed action}} \Big\} \;=\; 1+0 \overset{\checkmark}{\leq} 1 \;=\; r(s)\,,$$

and similar for z and t. As $r(s), r(t) < \infty$, we conclude $\mathbb{P}^{\min}_s(\Diamond T), \mathbb{P}^{\min}_t(\Diamond T) > 0$.

Remark 1 (Certificates for $\mathbb{P}^{\text{opt}}(\Diamond T) = 0$). While we do not need this in our paper, it is instructive to notice that with Lemmas 1 and 2 we can also certify *zero reachability probability*: By Knaster-Tarski, any r with $r \leq \mathcal{D}^{\overline{\text{opt}}}(r)$ witnesses $r \leq \operatorname{fp} \mathcal{D}^{\overline{\text{opt}}}$, hence if $r(s) = \infty$ for a state s, then $\mathbb{P}^{\text{opt}}_s(\Diamond T) = 0$.

Certificates for non-almost-sure (a.s.) reachability, i.e., $\mathbb{P}^{\text{opt}}(\Diamond T) < 1$, are needed in Section 5. Ranking function-based certificates for this property are—perhaps surprisingly—more involved. In Definition 2 below we define a *complementary distance operator* that captures (approximately) the distance to the states Z from which T is avoided surely, i.e., $\mathbb{P}^{\text{opt}}_s(\Diamond T) = 0$ for all $s \in Z$. By Lemma 2, finite distance to Z witnesses positive $\overline{\text{opt}}$-reachability of Z and thus non-a.s. opt-reachability of T. A major complication is that Z is not given explicitly. We address this by (i) considering the *least* fp, and (ii) letting the operator only increment the distance if two successors do not have the same rank. For this, we use *Iverson bracket* notation: $[\varphi] = 1$ if φ is true; $[\varphi] = 0$, else. Together, (i) and (ii) ensure that $s \in S$ has rank 0 in the lfp if and only if $s \in Z$.

Definition 2 (Complementary Distance Operator). *We define the* complementary distance operator $\overline{\mathcal{D}}^{\text{opt}} \colon \overline{\mathbb{N}}^S \to \overline{\mathbb{N}}^S$*, with*

$$\overline{\mathcal{D}}^{\text{opt}}(r)(s) \;=\; \begin{cases} \infty & \text{if } s \in T \\ \operatorname*{opt}_{a \in Act(s)} \Big(\min_{s' \in Post(s,a)} r(s') \;+\; \big[\exists u, v \in Post(s,a) \colon r(u) \neq r(v)\big] \Big) & \text{if } s \in S \setminus T \end{cases}$$

Note that unlike the distance operator $\mathcal{D}^{\text{opt}}$ from Definition 1, $\overline{\mathcal{D}}^{\text{opt}}$ *does not have a unique* fp: The constant $r = \boldsymbol{\infty}$ is always a trivial fixed point.

Lemma 3. *Let $r = \operatorname{lfp} \overline{\mathcal{D}}^{\text{opt}}$. Then for all $s \in S$, $r(s) = \infty \iff \mathbb{P}^{\text{opt}}_s(\Diamond T) = 1$.*

Proposition 2 (Certificates for $\mathbb{P}^{\text{opt}}(\Diamond T) < 1$). *A function $r \in \overline{\mathbb{N}}^S$ is called a* valid certificate for non-a.s. opt-reachability *if $\overline{\mathcal{D}}^{\text{opt}}(r) \leq r$. If r is valid, then $\forall s \in S \colon r(s) < \infty \implies \mathbb{P}^{\text{opt}}_s(\Diamond T) < 1$.*

Remark 2 (Certificates for $\mathbb{P}^{\text{opt}}(\Diamond T) = 1$). Since $\overline{\mathcal{D}}^{\text{opt}}$ does *not* have a unique fp, we cannot use the trick from Remark 1 to certify $\mathbb{P}^{\text{opt}}(\Diamond T) = 1$ with ranking functions. Sections 4.2 and 4.3 present certificates for general lower bounds.

4 Certificates for Quantitative Reachability

This section presents our certificates for bounds on minimal and maximal reachability probabilities (Table 1). They are characterized via a *Bellman operator*:

Definition 3 (Bellman Operator for Reachability). *We define the Bellman operator for reachability* $\mathcal{B}^{\text{opt}} \colon [0,1]^S \to [0,1]^S$ *as usual:*

$$\mathcal{B}^{\text{opt}}(x)(s) \;=\; \begin{cases} 1 & \text{if } s \in T \\ \underset{a \in Act(s)}{\text{opt}} \sum_{s' \in Post(s,a)} P(s,a,s') \cdot x(s') & \text{if } s \in S \setminus T \end{cases}$$

Similar to $\mathcal{D}^{\text{opt}}$ from Section 3, $\mathcal{B}^{\text{opt}}$ is a monotone function on the complete lattice $([0,1]^S, \leq)$. Thus, $\mathcal{B}^{\text{opt}}$ has a least fixed point by Theorem 1.

Theorem 2 ([16, Sec. 3.5]). *For all* $s \in S$, $(\text{lfp}\, \mathcal{B}^{\text{opt}})(s) = \mathbb{P}^{\text{opt}}_s(\Diamond T)$.

We stress that $\mathcal{B}^{\text{opt}}$ has multiple fixed points in general. For instance, $x = \mathbf{1}$ is always a trivial fixed point. Theorem 2 states that the reachability probabilities are characterized as the *least* fixed point.

4.1 Upper Bounds on Optimal Reachability Probabilities

Following Guiding Principle 1, we obtain the following by Theorem 2:

Proposition 3 (Certificates for Upper Bounds on $\mathbb{P}^{\text{opt}}(\Diamond T)$). *A probability vector* $x \in [0,1]^S$ *satisfying* $\mathcal{B}^{\text{opt}}(x) \leq x$ *is a* valid certificate for upper bounds on opt-reachability. *If* x *is valid, then* $\forall s \in S \colon \mathbb{P}^{\text{opt}}_s(\Diamond T) \leq x(s)$.

Example 2. We verify that the numbers x above the states in Figure 1 on page 2 are a valid certificate for upper bounds on min-reachability: For s we check

$$\mathcal{B}^{\min}(x)(s) \;=\; \min\left\{\tfrac{1}{3}\cdot 0 + \tfrac{1}{3}\cdot\tfrac{1}{2} + \tfrac{1}{3}\cdot 1 \;,\; 1\cdot 1\right\} \;=\; \min\left\{\tfrac{1}{2}, 1\right\} \overset{\checkmark}{\leq} \tfrac{1}{2} \;=\; x(s)\;,$$

and similar for z and t. Thus Proposition 3 yields $\mathbb{P}^{\min}_s(\Diamond T) \leq \frac{1}{2}$. This particular certificate remains valid when changing $x(s)$ to any probability in $[\frac{1}{2}, 1]$. In general, however, increasing individual values may break inductivity.

4.2 Lower Bounds on Minimal Reachability Probabilities

With Theorem 1, we can only certify lower bounds on *greatest* fixed points. Lower bounds on reachability probabilities—which constitute the *least* fixed point of $\mathcal{B}^{\text{opt}}$—are thus more involved. We propose to tackle this situation as follows:

Guiding Principle 3 (Modified Bellman Operators) *We often* modify *a basic Bellman-type operator to restrict its set of fixed points and enforce a certain extremal (i.e., least or greatest) fixed point of interest.*

We now focus on min-reachability first and modify $\mathcal{B}^{\min}$ as follows:

$$\tilde{\mathcal{B}}^{\min}\colon [0,1]^S \to [0,1]^S,\ \tilde{\mathcal{B}}^{\min}(x)(s) = \begin{cases} \mathcal{B}^{\min}(x)(s) & \text{if } \mathbb{P}^{\min}_s(\Diamond T) > 0 \\ 0 & \text{if } \mathbb{P}^{\min}_s(\Diamond T) = 0 \end{cases}$$

Lemma 4 (Unique Fixed Point [7, Thm. 10.109]). *$\tilde{\mathcal{B}}^{\min}$ has a unique fixed point* $\operatorname{fp} \tilde{\mathcal{B}}^{\min} = \operatorname{lfp} \mathcal{B}^{\min}$.

By Lemma 4 and Theorem 2, any probability vector $x \leq \tilde{\mathcal{B}}^{\min}(x)$ witnesses that $x(s) \leq \mathbb{P}^{\min}_s(\Diamond T)$ for all $s \in S$. However, evaluating $\tilde{\mathcal{B}}^{\min}(x)$ is not straightforward as it requires determining, for each $s \in S$ whether $\mathbb{P}^{\min}_s(\Diamond T) > 0$. Hence, we include an additional certificate for positive reachability from Section 3:

Proposition 4 (Certificates for Lower Bounds on $\mathbb{P}^{\min}(\Diamond T)$).
A tuple of probability vector and ranking function $(x, r) \in [0,1]^S \times \overline{\mathbb{N}}^S$ *is a* valid certificate for lower bounds on min-reachability *if*

$$1)\ \mathcal{D}^{\max}(r) \leq r, \quad 2)\ x \leq \mathcal{B}^{\min}(x), \quad 3)\ \forall s \in S \setminus T\colon x(s) > 0 \implies r(s) < \infty.$$

If (x, r) *is valid, then* $\forall s \in S\colon \mathbb{P}^{\min}_s(\Diamond T) \geq x(s)$.

Example 3. We apply Proposition 4 to the MDP in Figure 1. The pairs $x(v) \mid r(v)$ below each state v constitute a valid certificate (x, r) for lower bounds on min-reachability. Indeed, we have shown in Example 1 that it satisfies Condition 1) $\mathcal{D}^{\max}(r) \leq r$. Condition 2) $x \leq \mathcal{B}^{\min}(x)$ holds as well; in fact, we even have $x = \mathcal{B}^{\min}(x)$, see Example 2. For the additional Condition 3), notice that s is the only state in $S \setminus T$ with $x(s) > 0$, and that $r(s) < \infty$ holds as required. We conclude that $\mathbb{P}^{\min}_s(\Diamond T) \geq \frac{1}{2}$.

4.3 Lower Bounds on Maximal Reachability Probabilities

Our approach for lower bounds on $\mathbb{P}^{\min}(\Diamond T)$ from Section 4.2 does not immediately extend to max-reachability because $\tilde{\mathcal{B}}^{\max}$ (a modification of $\mathcal{B}^{\max}$ analogous to $\tilde{\mathcal{B}}^{\min}$) does *not* have a unique fixed point in general, see [17, App. D.3] for a concrete counterexample. This problem is caused by end components [2, Def. 3.13]. Towards a solution, we observe that, essentially by definition,

$$\forall s \in S\colon \quad \mathbb{P}^{\max}_s(\Diamond T) \geq x(s) \iff \exists \text{ Strategy } \sigma\colon \mathbb{P}^{\sigma}_s(\Diamond T) \geq x(s)\ .$$

In words, a lower bound on a max-reachability probability is always witnessed by some strategy.[5] Hence we adopt the following:

Guiding Principle 4 (Witness Strategies) *In some cases, especially when progress towards a target is required, it is helpful to certify a witness strategy.*

[5] Dually, an upper bound on a min-reachability probability is also witnessed by a strategy, but our corresponding certificates from Proposition 3 do not rely on this.

Certificates with an Explicit Witness Strategy. Recall from Section 2 that given a strategy $\sigma\colon S \to Act$ for MDP $\mathcal{M}$, we can consider the induced DTMC $\mathcal{M}^\sigma$. We write $\mathcal{B}^\sigma$ for the Bellman operator associated with $\mathcal{M}^\sigma$ (notice that a DTMC is just a special case of an MDP). Further, we let $\tilde{\mathcal{B}}^\sigma$ be the corresponding modified Bellman operator. By Theorem 2 and Lemma 4:

Lemma 5. *$\tilde{\mathcal{B}}^\sigma$ has a unique fixed point $(\mathrm{fp}\,\tilde{\mathcal{B}}^\sigma)(s) = \mathbb{P}^\sigma_s(\Diamond T)$ for all $s \in S$.*

Thus, we can certify lower bounds similar to Proposition 4 (we write $\mathcal{D}^\sigma$ for the distance operator $\mathcal{D}^{\mathrm{opt}}$ in the DTMC induced by σ):

Proposition 5 (Certificates for Lower Bounds on $\mathbb{P}^{\max}(\Diamond T)$ + Strategy). *A triple $(x, r, \sigma) \in [0,1]^S \times \overline{\mathbb{N}}^S \times Act^S$ is a* valid certificate for lower bounds on max-reachability with witness strategy *if*

$$1)\ \mathcal{D}^\sigma(r) \le r, \quad 2)\ x \le \mathcal{B}^\sigma(x), \quad 3)\ \forall s \in S \setminus T\colon x(s) > 0 \implies r(s) < \infty.$$

If (x, r, σ) is valid, then $\forall s \in S\colon \mathbb{P}^{\max}_s(\Diamond T) \ge \mathbb{P}^\sigma_s(\Diamond T) \ge x(s)$.

Certificates without a Witness Strategy. Increasing the size of the certificate by including the strategy can be avoided, as it can be "read off" from the certifying probability vector $x \in [0,1]^S$. To this end, we define the *x-increasing* actions of state $s \in S$: $Act^{\uparrow}_x(s) = \{a \in Act(s) \mid x(s) \le \sum_{s' \in Post(s,a)} P(s,a,s') \cdot x(s')\}$. If $x \le \mathcal{B}^{\max}(x)$, then $Act^{\uparrow}_x(s)$ contains at least one action. Next, we define a variant of the distance operator which only considers x-increasing actions:

$$\mathcal{D}^{\min}_{x\uparrow}\colon \overline{\mathbb{N}}^S \to \overline{\mathbb{N}}^S,\ \mathcal{D}^{\min}_{x\uparrow}(r)(s) = \begin{cases} 0 & \text{if } s \in T \\ 1 + \min\limits_{a \in Act^{\uparrow}_x(s)} \min\limits_{s' \in Post(s,a)} r(s') & \text{if } s \in S \setminus T \end{cases}$$

Proposition 6 (Certificates for Lower Bounds on $\mathbb{P}^{\max}(\Diamond T)$). *A tuple $(x, r) \in [0,1]^S \times \overline{\mathbb{N}}^S$ is a* valid certificate for lower bounds on max-reachability *if*

$$1)\ \mathcal{D}^{\min}_{x\uparrow}(r) \le r, \quad 2)\ x \le \mathcal{B}^{\max}(x), \quad 3)\ \forall s \in S \setminus T\colon x(s) > 0 \implies r(s) < \infty.$$

If (x, r) is valid, then $\forall s \in S\colon \mathbb{P}^{\max}_s(\Diamond T) \ge x(s)$.

5 Certificates for Expected Rewards

We present certificates for bounds on expected rewards (**??**) in the "$* = \infty$" semantics that assigns infinite reward to paths not reaching T, with the other case in [17, App. F]. We employ the reward variant of the Bellman operator:

Definition 4 (Bellman Operator for Expected Rewards). *We define the Bellman operator for expected rewards $\mathcal{E}^{\mathrm{opt}}\colon \overline{\mathbb{R}}^S_{\ge 0} \to \overline{\mathbb{R}}^S_{\ge 0}$ as follows:*

$$\mathcal{E}^{\mathrm{opt}}(x)(s) \;=\; \begin{cases} 0 & \text{if } s \in T \\ \mathrm{rew}(s) + \mathrm{opt}\limits_{a \in Act(s)} \sum\limits_{s' \in Post(s,a)} P(s,a,s') \cdot x(s') & \text{if } s \in S \setminus T \end{cases}$$

The above definition assumes that multiplication by ∞ *absorbs* positive numbers, i.e., $p \cdot \infty = \infty$ for all $p > 0$, and $a + \infty = \infty + a = \infty$ for all $a \in \overline{\mathbb{R}}_{\geq 0}$.

Again, $\mathcal{E}^{\text{opt}}$ is a monotone function on the complete lattice $(\overline{\mathbb{R}}^S_{\geq 0}, \leq)$ and thus has a least and a greatest fixed point by Theorem 1. Unfortunately, as it turns out, the sought-after expected rewards $\mathbb{E}^{\text{opt}}_s(\Diamond T)$, $s \in S$, are *neither of these two fixed points*. Indeed, $\operatorname{lfp} \mathcal{E}^{\text{opt}}$ corresponds to the expected rewards in the semantics considered in [17, App. F], and $\operatorname{gfp} \mathcal{E}^{\text{opt}}$ is a trivial upper bound assigning ∞ to all states, see the example in Section 5.1.

Remark 3 (Asymmetry and Duality). In Section 4, an asymmetry between upper and lower bounds arose as the reachability probabilities are a *least* fixed point. Further, for the case of maximizing reachability, spurious fixed points occurred and we required a witness strategy to "make progress" towards the targets (the fact that this case requires special treatment of end components is well established in literature, e.g., [31]). For *safety* objectives, where the goal is to avoid a set of bad states, the situation is dual: The safety probabilities are a *greatest* fixed point, so the lower bound case is simple, and when minimizing the upper bound, we require a witness strategy. The $* = \infty$ semantics for expected rewards share some similarities with a safety objective, since the value is maximized (i.e., is infinite) when the target set is avoided. This section thus differs from Section 4 in two ways: (i) Everything is dual, as $* = \infty$ is "safety-like", and (ii) additional complications arise from the trivial greatest fixed point $\operatorname{gfp} \mathcal{E}^{\text{opt}} = \boldsymbol{\infty}$, see below.

5.1 Lower Bounds on Optimal Expected Rewards

Due to the absorptive property of multiplication by ∞, $\operatorname{gfp} \mathcal{E}^{\text{opt}}$ may assign ∞ to states that actually have finite value: For instance, in the DTMC in Figure 2, the gfp assigns ∞ to s because $\infty = \operatorname{rew}(s) + \frac{1}{2} \cdot \infty + \frac{1}{2} \cdot 0$, while in fact $\mathbb{E}_s(\Diamond T) = 2 \cdot \operatorname{rew}(s) < \infty$. To address this, we force the values of states that a.s. reach the target to be finite as follows:

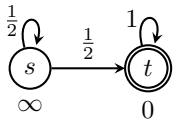


Fig. 2. A DTMC.

Lemma 6. *Let $x \in \overline{\mathbb{R}}^S_{\geq 0}$ be such that 1) $x \leq \mathcal{E}^{\text{opt}}(x)$ and 2) for all $s \in S$: $\mathbb{P}^{\overline{\text{opt}}}_s(\Diamond T) = 1 \implies x(s) < \infty$. Then it holds for all $s \in S$ that $x(s) \leq \mathbb{E}^{\text{opt}}_s(\Diamond T)$.*

Intuitively, Lemma 6 requires that a lower bound on $\mathbb{E}^{\min}_s(\Diamond T)$ can only be infinite if T cannot be reached a.s., i.e. $\mathbb{P}^{\max}_s(\Diamond T) < 1$ (dually for $\mathbb{E}^{\max}$). Combining Lemma 6 and a certificate for *non-a.s. reachability* (Section 3) yields:

Proposition 7 (Certificates for Lower Bounds on $\mathbb{E}^{\text{opt}}(\Diamond T)$). *A tuple $(x, r) \in \overline{\mathbb{R}}^S_{\geq 0} \times \overline{\mathbb{N}}^S$ is a* valid certificate for lower bounds on opt-exp. rewards *if*

$$1)\ \overline{\mathcal{D}^{\text{opt}}}(r) \leq r, \quad 2)\ x \leq \mathcal{E}^{\text{opt}}(x), \quad 3)\ \forall s \in S\colon x(s) = \infty \implies r(s) < \infty.$$

If (x, r) is valid, then $\forall s \in S\colon \mathbb{E}^{\text{opt}}_s(\Diamond T) \geq x(s)$.

5.2 Upper Bounds on Maximal Expected Rewards

Next we focus on upper bounds on maximal expected rewards. Using Guiding Principle 3 as for *lower* bounds on *minimal* reachability probabilities (Section 4.2), we obtain such certificates via a modified Bellman operator:

$$\tilde{\mathcal{E}}^{\max}\colon \overline{\mathbb{R}}^{S}_{\geq 0} \to \overline{\mathbb{R}}^{S}_{\geq 0}, \quad \tilde{\mathcal{E}}^{\max}(x)(s) \;=\; \begin{cases} \mathcal{E}^{\max}(x)(s) & \text{if } \mathbb{P}^{\min}_{s}(\Diamond T) > 0 \\ \infty & \text{if } \mathbb{P}^{\min}_{s}(\Diamond T) = 0 \end{cases}$$

Lemma 7. *For all* $s \in S$, $(\operatorname{lfp} \tilde{\mathcal{E}}^{\max})(s) = \mathbb{E}^{\max}_{s}(\Diamond T)$.

We stress that unlike $\tilde{\mathcal{B}}^{\min}$ from Section 4.2, $\tilde{\mathcal{E}}^{\max}$ does not have a *unique* fixed point, see Figure 2. Nonetheless, with Lemma 7, Guiding Principle 1, and the certificates for positive reachability from Proposition 1, we obtain:

Proposition 8 (Certificates for Upper Bounds on $\mathbb{E}^{\max}(\Diamond T)$). *A tuple* $(x, r) \in \overline{\mathbb{R}}^{S}_{\geq 0} \times \overline{\mathbb{N}}^{S}$ *is a* valid certificate for upper bounds on max-exp. rewards *if*

1) $\mathcal{D}^{\max}(r) \leq r$, 2) $\mathcal{E}^{\max}(x) \leq x$, 3) $\forall s \in S\colon x(s) < \infty \implies r(s) < \infty$.

If (x, r) *is valid, then* $\forall s \in S\colon \mathbb{E}^{\max}_{s}(\Diamond T) \leq x(s)$.

5.3 Upper Bounds on Minimal Expected Rewards

Our approach for this case parallels the one for *lower* bounds on *maximal* reachability probabilities from Section 4.3. The modified Bellman operator $\tilde{\mathcal{E}}^{\min}$ (defined analogous to $\tilde{\mathcal{E}}^{\max}$ from above) does *not* characterize the minimal expected rewards as its least fixed point. The problem are, again, end components, see [17, App. E.5] for a counter-example. Following Guiding Principle 4 and Section 4.3, we can, however, certify upper bounds on $\mathbb{E}^{\min}(\Diamond T)$ by including a witness strategy (see [17, App. E.6]).

As with lower bounds on max-reachability, it is also possible to avoid this explicit witness strategy: We define the *x-decreasing actions* of s as $Act^{\downarrow}_{x}(s) = \{a \in Act(s) \mid x(s) \geq \mathrm{rew}(s) + \sum_{s' \in Post(s,a)} P(s,a,s') \cdot x(s')\}$. If $\mathcal{E}^{\min}(x) \leq x$, then $Act^{\downarrow}_{x}(s) \neq \emptyset$. We define a distance operator with $\mathcal{D}^{\min}_{x\downarrow}$ that only considers x-decreasing actions completely analogous to $\mathcal{D}^{\min}_{x\uparrow}$ from Section 4.3.

Proposition 9 (Certificates for Upper Bounds on $\mathbb{E}^{\min}(\Diamond T)$). *A tuple* $(x, r) \in \overline{\mathbb{R}}^{S}_{\geq 0} \times \overline{\mathbb{N}}^{S}$ *is a* valid certificate for upper bounds on min-exp. rewards *if*

1) $\mathcal{D}^{\min}_{x\downarrow}(r) \leq r$, 2) $\mathcal{E}^{\min}(x) \leq x$, 3) $\forall s \in S\colon x(s) < \infty \implies r(s) < \infty$.

If (x, r) *is valid, then* $\forall s \in S\colon \mathbb{E}^{\min}_{s}(\Diamond T) \leq x(s)$.

6 Computing Certificates

In Sections 3 to 5 we described *what* certificates are and discussed their verification conditions. We now elaborate on *how to compute* certificates. To this

end, we first discuss computation of (co-)inductive value vectors x and then focus on the ranking functions r required by some certificates (see Table 1). We stress that a sound certificate checker detects any wrong results produced by buggy implementations of the methods discussed in this section. Indeed, during implementation of the certificate computation algorithms in Storm, checking the certificates helped finding and resolving implementation bugs.

As we enter the realm of numeric computation, some remarks are in order. For computational purposes we assume that the transitions probabilities are *rational numbers*, i.e., fractions of integers. Moreover:

Our goal is to compute a certificate with a rational value vector x and to check it with exact, arbitrary precision rational number arithmetic.

Certificates via Exact Algorithms. The conceptually easiest certifying MDP model checking algorithm is to compute the rational reachability probabilities or expected rewards *exactly*. The resulting value vector is both inductive and coinductive. Thus, exact algorithms yield a certificate essentially as a by-product. We refer to [29,30] for an in-depth comparison of exact algorithms based on *Policy Iteration* (PI), *Rational Search* (RS), and *Linear Programming* (LP). The practically most efficient algorithm is PI with exact LU decomposition as linear equation solver; see [30, Secs. 2.2 and 4.2] for a description of the algorithm.

Certificates via Approximate Algorithms. In practice, most probabilistic model checkers use algorithms that are not exact but approximate: They employ *approximate, fixed-precision floating point arithmetic* and use a variant of VI that only returns an approximate result, namely for each state an interval $[\ell, u]$ containing the exact value, such that $|\ell - u| \leq \varepsilon$ for a given ε (typically 10^{-6}). They do this because (i) when using exact arithmetic, fractions can grow very large, hindering scalability, (ii) VI-based algorithms often outperform PI, albeit not as dramatically as folklore claimed [29,30], and (iii) approximate results usually suffice. We now exemplify with the VI-variant *Interval Iteration* (II) [8,24] how to make an approximate, floating point-based algorithm certifying, leaving other variants such as *optimistic VI* [31] and *Sound VI* [49] for future work.

II for reachability[6] works by first *collapsing end components* [13,24] of the MDP to ensure that $\mathcal{B}^{\text{opt}}$ has a *unique* fixed point. II then runs two instances of VI in parallel, starting from $x^{(0)} = \mathbf{0}$ and $y^{(0)} = \mathbf{1}$:

$$\mathbf{0} = x^{(0)} \leq \mathcal{B}^{\text{opt}}(x^{(0)}) = x^{(1)} \leq \ \ldots \ \text{fp}\,\mathcal{B}^{\text{opt}} \ \ldots \ \leq y^{(1)} = \mathcal{B}^{\text{opt}}(y^{(0)}) \leq y^{(0)} = \mathbf{1}$$

Both sequences contain (co-)inductive vectors only and converge to the fixed point. The iteration can be stopped when the difference is as small as desired.

However, as we demonstrate experimentally (Section 7), *inexact floating point arithmetic usually breaks (co-)inductivity of the elements in the II sequences*, as was already reported in [55] in a similar setting. More precisely, let $\mathcal{B}^{\text{opt}}_{\mathbb{F}}$ be a "floating point variant" of $\mathcal{B}^{\text{opt}}$, i.e., the (exact) result of each operation is rounded *to a nearest float*. This the default rounding mode in IEEE 754. Let

[6] For expected rewards, II additionally requires computing an upper bound, see [8].

$x_{\mathbb{F}}^{(i)}$ be the i-th element, $i > 0$, in the lower VI sequence of $\mathcal{B}_{\mathbb{F}}^{\text{opt}}$ starting from $\mathbf{0}$. Then, due to rounding errors, $x_{\mathbb{F}}^{(i)} \leq \mathcal{B}^{\text{opt}}(x_{\mathbb{F}}^{(i)})$ does *not* hold in general, i.e., $x_{\mathbb{F}}^{(i)}$ might not be co-inductive. We propose two ways to mitigate this problem: *Safe rounding* [27] and *Smooth II*.

First, safe rounding amounts to configuring the IEEE754 rounding mode so that results of floating point computations are always rounded towards 0 when iterating from below, and towards ∞ when iterating from above. While safe rounding provably yields sound bounds [27], it may slow down or even prevent convergence of II. Nonetheless, in practice, II with safe rounding finds significantly more certificates than II with default rounding (Section 7).

Second, for Smooth II we define the γ-*smooth Bellman operator* ($\gamma \in [0, 1)$)

$$\mathcal{B}_{\gamma}^{\text{opt}}(x) \;=\; \gamma \cdot x + (1 - \gamma) \cdot \mathcal{B}^{\text{opt}}(x) \;,$$

where scalar multiplication and addition are component-wise. $\mathcal{B}_{\gamma}^{\text{opt}}$ and $\mathcal{B}^{\text{opt}}$ have the same fixed points, and every (co-)inductive value vector w.r.t. $\mathcal{B}_{\gamma}^{\text{opt}}$ is also (co-)inductive w.r.t. $\mathcal{B}^{\text{opt}}$ [17, App. G.1]. The key property of $\mathcal{B}_{\gamma}^{\text{opt}}$ compared to $\mathcal{B}^{\text{opt}}$ is that the former enforces *ultimately strictly monotonic VI sequences*. This mitigates the floating point rounding issues. Notice, however, that smoothing slows down convergence. Smoothing and safe rounding may be combined.

Computing Ranking Functions. We briefly outline how to obtain the unique and least fixed points of $\mathcal{D}^{\text{opt}}$ and $\overline{\mathcal{D}}^{\text{opt}}$, respectively (see Definitions 1 and 2).

First, $\text{fp}\,\mathcal{D}^{\text{opt}}$ can be computed via VI from $r^{(0)} = \boldsymbol{\infty}$. This iteration converges in finitely many steps. Second, to compute $\text{lfp}\,\overline{\mathcal{D}}^{\text{opt}}$ we propose to perform VI from $r^{(0)}$ with $r^{(0)}(s) = [\mathbb{P}_s^{\overline{\text{opt}}}(\Diamond T) = 1] \cdot \infty$ for all $s \in S$. The condition in the Iverson bracket can be evaluated using standard graph analysis [7, Section 10.6.1]. This iteration converges in finitely many steps as well, see [17, Apps. G.2 and G.3] for details and a practically more efficient algorithm.

7 Experimental Evaluation

Implementation. We implemented certificate computation as discussed in Section 6 in Storm [33]. Given a higher-level model description (PRISM [42] or Jani [14]), and a reachability probability or expected reward query, our implementation proceeds in three steps: First, Storm builds an explicit MDP from the description. Second, it computes a certificate for both lower *and* upper bounds, such that the relative difference between the two values is at most $\varepsilon = 10^{-6}$ for each MDP state. Finally, Storm checks the validity of the certificate.

Following the discussion in Section 6, we consider the following algorithms: Regarding exact computation, we use PI with *exact* LU decomposition, called PI^X. For approximate computation with floating point arithmetic, we employ II. Further, to investigate the impact of the rounding error mitigation techniques from Section 6, we complement II with either safe rounding (denoted II_{rnd}), smoothing with parameter γ (denoted $\text{II}^{\circlearrowleft\gamma}$; we consider $\gamma \in \{0.05, 0.8, 0.9, 0.95\}$), or a combination of both (denoted $\text{II}_{\text{rnd}}^{\circlearrowleft\gamma}$). Overall, we compare PI^X and seven

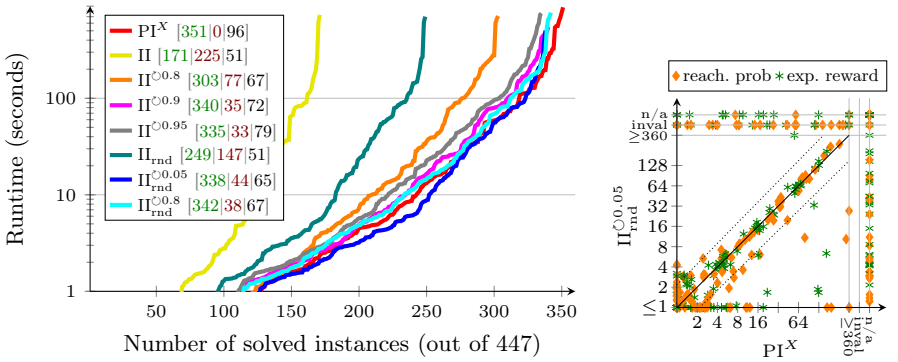


Fig. 3. RQ1: Runtime for computing certificates of PI^X and several combinations of mitigation techniques with II (left); and detailed comparison of PI^X and $\mathrm{II}_{\mathrm{rnd}}^{\circlearrowleft 0.05}$ (right).

variants of II. We employ additional standard modifications of the algorithms, namely: We eliminate end components whenever possible, apply topological optimizations for PI^X and II, and apply Gauß-Seidl Bellman updates for II [8,29].

In all three steps of the implementation, we represent numbers as arbitrary precision rationals implemented in GMP [23]—except when running II in the second step (in which case we convert rationals to their nearest floats, potentially yielding invalid certificates). We thus certify reachability probabilities and expected rewards with respect to the *exact* MDP without rounding errors.

The MDP and the certificate computed with Storm can be exported and checked by an independent *formally verified certificate checker*. To construct the latter, we verified the correctness of the certificate checking algorithms in the interactive proof assistant Isabelle/HOL [46], extending previous work on MDPs [34,50] by total rewards and qualitative reachability properties. Based on this library, we proved correct the soundness of the certificates described in Sections 3 to 5. We used Isabelle/HOL's code export mechanism [25] to obtain a verified, executable Standard ML implementation that employs exact rational arithmetic. The construction of the MDP from a PRISM or Jani model as well as export and parsing of MDPs and certificates are currently not verified.

Benchmarks and Setup. We use all 366 benchmark instances from the quantitative verification benchmark set (QVBS) [32] that (i) consider an MDP with a reachability or reward objective and (ii) for which Storm can build an explicit representation within 5 minutes. Additionally, since the QVBS contains no models exhibiting non-trivial ECs, we include 71 structurally diverse models from various sources detailed in [30, Sec. 5.3]. Overall, we consider the complete *alljani* set from [30]. We invoke Storm for each combination of benchmark instance and certificate algorithm and report the overall runtime (walltime). All experiments ran on Intel Xeon 8468 Sapphire 2.1 Ghz systems. We used Slurm to limit the individual executions to 4 CPU cores and 16 GB of RAM, with a time limit of 900 s. Next, we discuss our findings by answering three research questions (RQs).

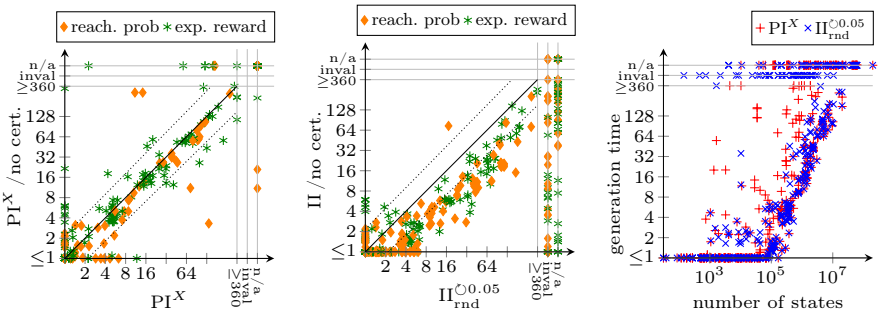


Fig. 4. RQ2: Runtime overhead of certified MDP model checking for PI^X (left) and II (middle), and scalability of both with respect to the number of states (right).

RQ1: Best algorithm for certificate generation? Figure 3 (left) compares the runtimes of PI^X and our seven II variants. A point (x, y) for algorithm A indicates that there are x instances for which A computes a *valid* certificate within y seconds (including time for model construction but excluding time for exporting the certificate files). The triples $|v|w|u|$ in the legend indicate that the algorithm produced a total of v valid and w invalid certificates (with invalidity likely due to floating point issues), while for the remaining u instances no result was found within the resource limits. As expected, all certificates produced by the exact PI^X are valid, while standard II produces many invalid certificates. Safe rounding and smoothing improve the number of valid certificates. Notably, $\mathrm{II}^{\circlearrowleft\gamma}$ (only smoothing) performs best for γ values close to 1, while the performance of $\mathrm{II}^{\circlearrowleft\gamma}_{\mathrm{rnd}}$ (smoothing *and* safe rounding) is less sensitive towards γ; see [17, App. H] for more details. Among all II variants, $\mathrm{II}^{\circlearrowleft 0.05}_{\mathrm{rnd}}$ shows the best overall performance.

The scatter plot in Figure 3 (right) further compares PI^X and $\mathrm{II}^{\circlearrowleft 0.05}_{\mathrm{rnd}}$. A data point (x, y) corresponds to one benchmark instance, where x and y are runtimes of PI^X and $\mathrm{II}^{\circlearrowleft 0.05}_{\mathrm{rnd}}$. A point at ≥ 300 indicates a runtime between 300 and 900 seconds, *inval* means an invalid certificate, and *n/a* denotes an aborted computation due to time/memory limits. Many instances that PI^X cannot solve are solved by $\mathrm{II}^{\circlearrowleft 0.05}_{\mathrm{rnd}}$ and vice versa. This is already the case without computing certificates, as the structure of a benchmark affects the performance of the algorithms differently [29,30]. Thus, as in the case without computing certificates, there is no "best algorithm", and both PI^X and variants of II can be considered. Overall, 396 out of 447 instances are correctly solved and certified by PI^X or $\mathrm{II}^{\circlearrowleft 0.05}_{\mathrm{rnd}}$ (or both). We highlight that PI^X is not only a complete certifying algorithm, but also practically efficient, even though it uses exact arithmetic.

RQ2: Runtime overhead of certificate generation? Figure 4 (left/middle) reports the runtime overhead of generating a certificate for PI^X and $\mathrm{II}^{\circlearrowleft 0.05}_{\mathrm{rnd}}$. For PI^X, the overhead is typically within a factor of 2, often significantly less. It is sometimes faster due to implementation differences in the certifying variant of PI^X. For $\mathrm{II}^{\circlearrowleft 0.05}_{\mathrm{rnd}}$, the overhead is slightly larger, typically around 1.5 to 4. This is partly due to the slower convergence caused by smoothing. Figure 4 (right) investigates

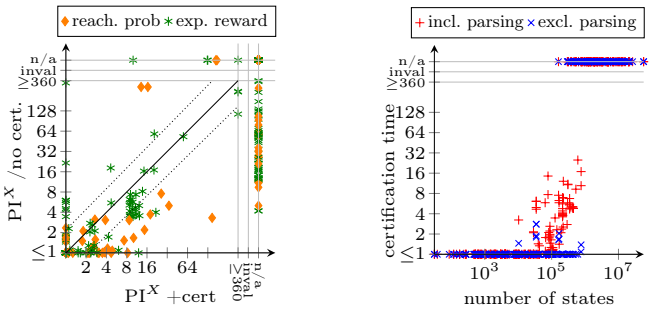


Fig. 5. RQ3: Runtime overhead of the certified pipeline/Runtime of certificate checking

the scalability of certificate generation with respect to the number of states. For MDPs with up to 10^5 states, certificate generation usually completes within a minute (often much less); for more than 10^7 states, it usually times out.

RQ3: Scalability of the formally verified certificate checker? Figure 5 (left) compares the runtime of the full pipeline including certificate generation and verification using our formally verified checker (PI^X +cert) with plain, uncertified MDP model checking based on PI^X. Compared to Figure 4 (left), the added verification of the certificates causes additional time/memory outs, and roughly doubles the runtime of the other instances. Figure 5 (right) reveals that parsing is currently a major bottleneck in the verified checker. Nonetheless, the checker completes within a few seconds on MDPs with up to $\approx 10^5$ states, and usually within 30 s for instances with up to $\approx 10^6$ states.

8 Conclusion and Future Work

We proposed *fixed point certificates* as a new standard for certified model checking of reachability and expected reward properties in MDPs. The soundness of these certificates was formalized in Isabelle/HOL, increasing their trustworthiness and enabling us to generate a formally verified certificate checker, applicable to non-trivial practically relevant instances. Our certificates can be generated with moderate overhead via minor, yet careful, modifications of established algorithms like II or PI. This allows tool developers and competitions [15]—for which our certificates provide formally verified reference results—to adopt our proposal with relatively low effort. Future work is to develop a more efficient certificate format. Further, we plan to extend our theory to other quantitative verification settings [3], e.g., stochastic games and ω-regular properties, and make it amenable to techniques such as *symbolic model checking* and *partial exploration*.

Data availability statement. The models, tools, and scripts to reproduce our experimental evaluation are available at DOI 10.5281/zenodo.14626585 [18].

References

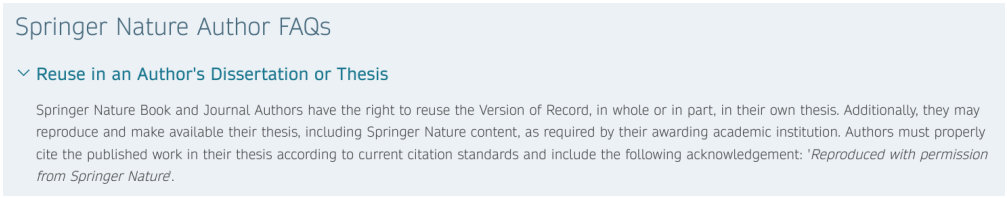
1. Abate, A., Giacobbe, M., Roy, D.: Stochastic omega-regular verification and control with supermartingales. In: CAV (3). Lecture Notes in Computer Science, vol. 14683, pp. 395–419. Springer (2024). https://doi.org/10.1007/978-3-031-65633-0_18
2. de Alfaro, L.: Formal verification of probabilistic systems. Ph.D. thesis, Stanford University, USA (1997)
3. Andriushchenko, R., Bork, A., Budde, C.E., Češka, M., Hahn, E.M., Hartmanns, A., Israelsen, B., Jansen, N., Jeppson, J., Junges, S., Köhl, M.A., Könighofer, B., Křetínský, J., Meggendorfer, T., Parker, D., Pranger, S., Quatmann, T., Ruijters, E., Taylor, L., Volk, M., Weininger, M., Zhang, Z.: Tools at the frontiers of quantitative verification: QComp 2023 competition report. In: International TOOLympics Challenge, pp. 90–146. Springer (2024)
4. Azeem, M., Evangelidis, A., Kretínský, J., Slivinskiy, A., Weininger, M.: Optimistic and topological value iteration for simple stochastic games. In: ATVA. Lecture Notes in Computer Science, vol. 13505, pp. 285–302. Springer (2022). https://doi.org/10.1007/978-3-031-19992-9_18
5. Baier, C., de Alfaro, L., Forejt, V., Kwiatkowska, M.: Model checking probabilistic systems. In: Handbook of Model Checking, pp. 963–999. Springer (2018). https://doi.org/10.1007/978-3-319-10575-8_28
6. Baier, C., Chau, C., Klüppelholz, S.: Certificates and witnesses for multi-objective queries in Markov decision processes. In: QEST+FORMATS. Lecture Notes in Computer Science, vol. 14996, pp. 1–18. Springer (2024). https://doi.org/10.1007/978-3-031-68416-6_1
7. Baier, C., Katoen, J.: Principles of model checking. MIT Press (2008)
8. Baier, C., Klein, J., Leuschner, L., Parker, D., Wunderlich, S.: Ensuring the reliability of your model checker: Interval iteration for Markov decision processes. In: CAV (1). Lecture Notes in Computer Science, vol. 10426, pp. 160–180. Springer (2017). https://doi.org/10.1007/978-3-319-63387-9_8
9. Balyo, T., Heule, M., Iser, M., Järvisalo, M., Suda, M.: Proceedings of SAT Competition 2023: Solver, benchmark and proof checker descriptions (2023), https://researchportal.helsinki.fi/en/publications/proceedings-of-sat-competition-2023-solver-benchmark-and-proof-ch
10. Batz, K., Biskup, T.J., Katoen, J., Winkler, T.: Programmatic strategy synthesis: Resolving nondeterminism in probabilistic programs. Proc. ACM Program. Lang. **8**(POPL), 2792–2820 (2024). https://doi.org/10.1145/3632935
11. Bellman, R.: Dynamic programming and stochastic control processes. Inf. Control. **1**(3), 228–239 (1958). https://doi.org/10.1016/S0019-9958(58)80003-0
12. Beyer, D.: Competition on software verification and witness validation: SV-COMP 2023. In: TACAS (2). Lecture Notes in Computer Science, vol. 13994, pp. 495–522. Springer (2023). https://doi.org/10.1007/978-3-031-30820-8_29
13. Brázdil, T., Chatterjee, K., Chmelik, M., Forejt, V., Kretínský, J., Kwiatkowska, M.Z., Parker, D., Ujma, M.: Verification of Markov decision processes using learning algorithms. In: ATVA. Lecture Notes in Computer Science, vol. 8837, pp. 98–114. Springer (2014). https://doi.org/10.1007/978-3-319-11936-6_8
14. Budde, C.E., Dehnert, C., Hahn, E.M., Hartmanns, A., Junges, S., Turrini, A.: JANI: quantitative model and tool interaction. In: TACAS (2). Lecture Notes in Computer Science, vol. 10206, pp. 151–168 (2017). https://doi.org/10.1007/978-3-662-54580-5_9

15. Budde, C.E., Hartmanns, A., Klauck, M., Kretínský, J., Parker, D., Quatmann, T., Turrini, A., Zhang, Z.: On correctness, precision, and performance in quantitative verification - QComp 2020 competition report. In: ISoLA (4). Lecture Notes in Computer Science, vol. 12479, pp. 216–241. Springer (2020). https://doi.org/10.1007/978-3-030-83723-5_15
16. Chatterjee, K., Henzinger, T.A.: Value iteration. In: 25 Years of Model Checking. Lecture Notes in Computer Science, vol. 5000, pp. 107–138. Springer (2008). https://doi.org/10.1007/978-3-540-69850-0_7
17. Chatterjee, K., Quatmann, T., Schäffeler, M., Weininger, M., Winkler, T., Zilken, D.: Fixed point certificates for reachability and expected rewards in MDPs. CoRR **abs/2501.11467** (2025), https://arxiv.org/abs/2501.11467
18. Chatterjee, K., Quatmann, T., Schäffeler, M., Weininger, M., Winkler, T., Zillken, D.: Artifact: Fixed point certificates for reachability and expected rewards in mdps (2025). https://doi.org/10.5281/zenodo.14626585
19. Chen, T., Forejt, V., Kwiatkowska, M.Z., Parker, D., Simaitis, A.: Automatic verification of competitive stochastic systems. Formal Methods Syst. Des. **43**(1), 61–92 (2013). https://doi.org/10.1007/S10703-013-0183-7, https://doi.org/10.1007/s10703-013-0183-7
20. Debbi, H.: Counterexamples in model checking - A survey. Informatica (Slovenia) **42**(2) (2018), http://www.informatica.si/index.php/informatica/article/view/1442
21. Froleyks, N., Yu, E., Biere, A., Heljanko, K.: Certifying phase abstraction. In: IJCAR (1). Lecture Notes in Computer Science, vol. 14739, pp. 284–303. Springer (2024). https://doi.org/10.1007/978-3-031-63498-7_17
22. Funke, F., Jantsch, S., Baier, C.: Farkas certificates and minimal witnesses for probabilistic reachability constraints. In: TACAS (1). Lecture Notes in Computer Science, vol. 12078, pp. 324–345. Springer (2020). https://doi.org/10.1007/978-3-030-45190-5_18
23. Granlund, T., Team, G.D.: GNU MP 6.0 Multiple Precision Arithmetic Library. Samurai Media Limited (2015)
24. Haddad, S., Monmege, B.: Interval iteration algorithm for MDPs and IMDPs. Theor. Comput. Sci. **735**, 111–131 (2018). https://doi.org/10.1016/J.TCS.2016.12.003
25. Haftmann, F., Krauss, A., Kuncar, O., Nipkow, T.: Data refinement in Isabelle/HOL. In: ITP. Lecture Notes in Computer Science, vol. 7998, pp. 100–115. Springer (2013). https://doi.org/10.1007/978-3-642-39634-2_10
26. Hahn, E.M., Hartmanns, A., Hensel, C., Klauck, M., Klein, J., Kretínský, J., Parker, D., Quatmann, T., Ruijters, E., Steinmetz, M.: The 2019 comparison of tools for the analysis of quantitative formal models - (QComp 2019 competition report). In: TACAS (3). Lecture Notes in Computer Science, vol. 11429, pp. 69–92. Springer (2019). https://doi.org/10.1007/978-3-030-17502-3_5
27. Hartmanns, A.: Correct probabilistic model checking with floating-point arithmetic. In: TACAS (2). Lecture Notes in Computer Science, vol. 13244, pp. 41–59. Springer (2022). https://doi.org/10.1007/978-3-030-99527-0_3
28. Hartmanns, A., Hermanns, H.: The Modest Toolset: An integrated environment for quantitative modelling and verification. In: TACAS. LNCS, vol. 8413, pp. 593–598. Springer (2014). https://doi.org/10.1007/978-3-642-54862-8_51
29. Hartmanns, A., Junges, S., Quatmann, T., Weininger, M.: A practitioner's guide to MDP model checking algorithms. In: TACAS (1). Lecture Notes in Computer Science, vol. 13993, pp. 469–488. Springer (2023). https://doi.org/10.1007/978-3-031-30823-9_24

30. Hartmanns, A., Junges, S., Quatmann, T., Weininger, M.: The revised practitioner's guide to MDP model checking algorithms. Under submission, Preprint: https://sjunges.github.io/files/revised_practitioners_guide.pdf (2025)
31. Hartmanns, A., Kaminski, B.L.: Optimistic value iteration. In: CAV (2). Lecture Notes in Computer Science, vol. 12225, pp. 488–511. Springer (2020). https://doi.org/10.1007/978-3-030-53291-8_26
32. Hartmanns, A., Klauck, M., Parker, D., Quatmann, T., Ruijters, E.: The quantitative verification benchmark set. In: TACAS (1). Lecture Notes in Computer Science, vol. 11427, pp. 344–350. Springer (2019). https://doi.org/10.1007/978-3-030-17462-0_20
33. Hensel, C., Junges, S., Katoen, J., Quatmann, T., Volk, M.: The probabilistic model checker Storm. Int. J. Softw. Tools Technol. Transf. **24**(4), 589–610 (2022). https://doi.org/10.1007/S10009-021-00633-Z
34. Hölzl, J.: Markov chains and Markov decision processes in Isabelle/HOL. J. Autom. Reason. **59**(3), 345–387 (2017). https://doi.org/10.1007/S10817-016-9401-5
35. Jantsch, S.: Certificates and Witnesses for Probabilistic Model Checking. Ph.D. thesis, Dresden University of Technology, Germany (2022)
36. Jantsch, S., Harder, H., Funke, F., Baier, C.: SWITSS: Computing small witnessing subsystems. In: FMCAD. pp. 236–244. IEEE (2020). https://doi.org/10.34727/2020/ISBN.978-3-85448-042-6_31
37. Junges, S., Katoen, J., Pérez, G.A., Winkler, T.: The complexity of reachability in parametric Markov decision processes. J. Comput. Syst. Sci. **119**, 183–210 (2021). https://doi.org/10.1016/J.JCSS.2021.02.006
38. Kratsch, D., McConnell, R.M., Mehlhorn, K., Spinrad, J.P.: Certifying algorithms for recognizing interval graphs and permutation graphs. SIAM J. Comput. **36**(2), 326–353 (2006). https://doi.org/10.1137/S0097539703437855
39. Kretínský, J., Meggendorfer, T., Weininger, M.: Stopping criteria for value iteration on stochastic games with quantitative objectives. In: LICS. pp. 1–14. IEEE (2023). https://doi.org/10.1109/LICS56636.2023.10175771
40. Kupferman, O., Sickert, S.: Certifying inexpressibility. In: FoSSaCS. Lecture Notes in Computer Science, vol. 12650, pp. 385–405. Springer (2021). https://doi.org/10.1007/978-3-030-71995-1_20
41. Kupferman, O., Vardi, M.Y.: From complementation to certification. Theor. Comput. Sci. **345**(1), 83–100 (2005). https://doi.org/10.1016/J.TCS.2005.07.021
42. Kwiatkowska, M.Z., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: CAV. Lecture Notes in Computer Science, vol. 6806, pp. 585–591. Springer (2011). https://doi.org/10.1007/978-3-642-22110-1_47
43. Lechner, M., Zikelic, D., Chatterjee, K., Henzinger, T.A.: Stability verification in stochastic control systems via neural network supermartingales. In: AAAI. pp. 7326–7336. AAAI Press (2022). https://doi.org/10.1609/AAAI.V36I7.20695
44. McConnell, R.M., Mehlhorn, K., Näher, S., Schweitzer, P.: Certifying algorithms. Comput. Sci. Rev. **5**(2), 119–161 (2011). https://doi.org/10.1016/J.COSREV.2010.09.009
45. Namjoshi, K.S.: Certifying model checkers. In: CAV. Lecture Notes in Computer Science, vol. 2102, pp. 2–13. Springer (2001). https://doi.org/10.1007/3-540-44585-4_2
46. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL - A Proof Assistant for Higher-Order Logic, Lecture Notes in Computer Science, vol. 2283. Springer (2002). https://doi.org/10.1007/3-540-45949-9, https://doi.org/10.1007/3-540-45949-9

47. Peled, D.A., Pnueli, A., Zuck, L.D.: From falsification to verification. In: FSTTCS. Lecture Notes in Computer Science, vol. 2245, pp. 292–304. Springer (2001). https://doi.org/10.1007/3-540-45294-X_25
48. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley Series in Probability and Statistics, Wiley (1994). https://doi.org/10.1002/9780470316887
49. Quatmann, T., Katoen, J.: Sound value iteration. In: CAV (1). Lecture Notes in Computer Science, vol. 10981, pp. 643–661. Springer (2018). https://doi.org/10.1007/978-3-319-96145-3_37
50. Schäffeler, M., Abdulaziz, M.: Formally verified solution methods for Markov decision processes. In: AAAI. pp. 15073–15081. AAAI Press (2023). https://doi.org/10.1609/AAAI.V37I12.26759
51. Takisaka, T., Zhang, L., Wang, C., Liu, J.: Lexicographic ranking supermartingales with lazy lower bounds. In: CAV (3). Lecture Notes in Computer Science, vol. 14683, pp. 420–442. Springer (2024). https://doi.org/10.1007/978-3-031-65633-0_19
52. Tan, Y.K., Yang, J., Soos, M., Myreen, M.O., Meel, K.S.: Formally certified approximate model counting. In: CAV (1). Lecture Notes in Computer Science, vol. 14681, pp. 153–177. Springer (2024). https://doi.org/10.1007/978-3-031-65627-9_8
53. White, D.J.: A survey of applications of Markov decision processes. Journal of the operational research society **44**(11), 1073–1096 (1993). https://doi.org/10.1057/jors.1993.181
54. Wimmer, R., Jansen, N., Ábrahám, E., Katoen, J., Becker, B.: Minimal counterexamples for linear-time probabilistic verification. Theor. Comput. Sci. **549**, 61–100 (2014). https://doi.org/10.1016/J.TCS.2014.06.020, https://doi.org/10.1016/j.tcs.2014.06.020
55. Winkler, T., Katoen, J.: Certificates for probabilistic pushdown automata via optimistic value iteration. In: TACAS (2). Lecture Notes in Computer Science, vol. 13994, pp. 391–409. Springer (2023). https://doi.org/10.1007/978-3-031-30820-8_24
56. Winkler, T., Katoen, J.: On certificates, expected runtimes, and termination in probabilistic pushdown automata. In: LICS. pp. 1–13. IEEE (2023). https://doi.org/10.1109/LICS56636.2023.10175714
57. Yu, E., Biere, A., Heljanko, K.: Progress in certifying hardware model checking results. In: CAV (2). Lecture Notes in Computer Science, vol. 12760, pp. 363–386. Springer (2021). https://doi.org/10.1007/978-3-030-81688-9_17
58. Yu, E., Froleyks, N., Biere, A., Heljanko, K.: Stratified certification for k-induction. In: FMCAD. pp. 59–64. IEEE (2022). https://doi.org/10.34727/2022/ISBN.978-3-85448-053-2_11
59. Yu, E., Froleyks, N., Biere, A., Heljanko, K.: Towards compositional hardware model checking certification. In: FMCAD. pp. 1–11. IEEE (2023). https://doi.org/10.34727/2023/ISBN.978-3-85448-060-0_12

Copyright and License Information


Springer Nature Author FAQs

Reuse in an Author's Dissertation or Thesis

Springer Nature Book and Journal Authors have the right to reuse the Version of Record, in whole or in part, in their own thesis. Additionally, they may reproduce and make available their thesis, including Springer Nature content, as required by their awarding academic institution. Authors must properly cite the published work in their thesis according to current citation standards and include the following acknowledgement: '*Reproduced with permission from Springer Nature*'.

Figure 2: Springer Nature permission, retrieved from `https://www.springernature.com/gp/partners/rights-permissions-third-party-distribution` (accessed on April 10, 2025)