Fixed-Point Reasoning for Stochastic Systems
A Survey of Recent Advancements and Open Challenges

Zhiyang Li[1], Mingqi Yang[1], Shenghua Feng[2(✉)], and Mingshuai Chen[1(✉)]

[1] Zhejiang University, Hangzhou, China
`{misakalzy,mingqiyang,m.chen}@zju.edu.cn`
[2] Zhongguancun Laboratory, Beijing, China
`fengsh@zgclab.edu.cn`

Abstract. Fixed points are mathematical objects that are commonly employed in computer science to characterize key properties of iterative or cyclic behaviors, e.g., unbounded loops and recursions in programs or cycles in transition systems. Reasoning about such behaviors is arguably the hardest task in formal verification. The problem is even harder for stochastic systems as it often amounts to inferring quantitative fixed points that are highly intractable in practice. This article surveys recent advancements in fixed-point reasoning for stochastic systems modeled as probabilistic programs, probabilistic transition systems, and stochastic hybrid systems and outlines potential directions for future research thereof. The core of our results is the *fixed-point reasoning landscape for stochastic systems*, where we focus on formal techniques that either (i) establish sound over-/under-approximations of quantitative fixed points; or (ii) infer the exact fixed points for a restricted class of systems.

Keywords: Quantitative verification · Fixed-point reasoning · Stochastic systems · Markov models

1 Introduction

A *fixed point* [70] is a quantity that remains unchanged under a given transformation. In the theory of computation, it represents stability and self-reference, as is the heart of recursion and iterative processes [90,93,1]. The employment of fixed points in (theoretical) computer science was initiated by Scott in the late 1960s to establish denotational semantics for λ-calculus [92]. Since then, the theory of fixed points has witnessed various applications in characterizing key properties of *looping behaviors* central to various computational models: The least or greatest fixed point (in terms of a partial order) intrinsically represents the set of reachable states in a transition system [4], the winning strategy of an ω-regular game [87], and – in the stochastic setting – the stationary distribution of a Markov chain [85] as well as the denotational semantics of a potentially unbounded loop in probabilistic programs [65,75,56].

Computing the exact (least or greatest) fixed point is, nonetheless, highly intractable or even impossible in practice, as it often requires an infinite or, in some cases, transfinite number of fixed-point iterations (see Section 2.2). The problem is known to be harder for stochastic systems (see, e.g., [57]) where *quantitative properties* are of particular interest, such as probabilities, expected values, high-order moments, and concentrations. Consequently, existing techniques aim to either (i) infer the exact quantitative fixed points for a restricted subclass of systems; or (ii) establish sound over-/under-approximations of fixed points.

In this article, we sketch the *quantitative fixed-point reasoning landscape for stochastic systems*. This landscape serves as a taxonomy of core formal techniques for *verifying*, *refuting*, or *synthesizing* upper bounds (over-approximations), lower bounds (under-approximations), or exact values for the least (or, dually, greatest) fixed point. On top of the landscape, we survey recent advancements (including our own results) in fixed-point reasoning for stochastic systems modeled as probabilistic programs [64,44,76,11], probabilistic transition systems (more specifically, Markov models [4,35]), and stochastic hybrid systems [51,24]. We further identify a few initial attempts that aim to establish a unified fixed-point reasoning framework for (part of) these system models. Based on the landscape, we outline several open challenges and potential research directions.

Remark 1. The work surveyed in this article is far from being exhaustive considering especially the actively developing field of quantitative verification; our goal is to provide interested readers with a bunch of pointers to navigate through the rudimentary fixed-point reasoning landscape for stochastic systems. $\lhd$

Paper structure. The rest of the paper is organized as follows. Section 2 recaps mathematical foundations of fixed points with a main focus on established results in domain theory. Section 3 presents our quantitative fixed-point reasoning landscape for stochastic systems, which is subsequently elaborated in Section 4 for probabilistic programs, Section 5 for Markov models, and Section 6 for stochastic hybrid systems. We then survey in Section 7 research attempts for providing unified fixed-point reasoning frameworks across different system models. In Section 8, we outline open challenges and potential directions for future research. The paper is finally concluded in Section 9.

2 Mathematical Foundations of Fixed Points

This section formulates – in the realm of domain theory [93,1] – the fixed-point reasoning problems and summarizes general reasoning techniques underpinning the (quantitative) verification of iterative (or cyclic) behaviors in computer science. It is briefly complemented by an alternative, well-established perspective on fixed points in metric spaces [2] for formulating key properties of MDPs.

2.1 Fixed-Point Reasoning Problems

Intuitively, a *fixed point* of a function $f\colon X \to Y$ is an element $x \in X \cap Y$ that is mapped to itself by the function, namely,

$$f(x) \;=\; x\ .$$

As a simple example, the real-valued function[3] $\lambda x\boldsymbol{.}\; x^3 - x^2 - x + 2$ admits multiple fixed points $x = -\sqrt{2}$, 1, and $\sqrt{2}$ as depicted in Fig. 1.

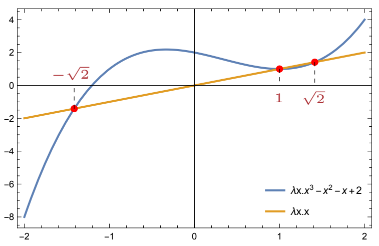


Fig. 1: Examples of fixed points.

Fixed points are essential to computer science as they provide a neat mechanism to characterize key properties of *looping behaviors* central to various computational models, ranging from programs to state-transition systems: Their self-reference nature – where the process revisits the same computational steps under varying states – makes it particularly challenging for formally analyzing such behaviors. Taking a (possibly unbounded) program loop as an example, the technique of structural induction [102] can be used to precisely characterize the loop's semantics as a certain fixed point of the so-called characteristic function – a monotonic operator mimicking the effect of unfolding the loop (more technical details can be found in Section 4). Such characterization aids in understanding and certifying key properties of looping behaviors, such as termination and correctness across diverse computational contexts. For instance, fixed points can also be employed to encode the set of reachable states in a transition system [4] and the stationary distribution of a Markov chain [85].

Below, we formalize several key problems in fixed-point reasoning leveraging well-established tools in domain theory initiated by Scott in the late 1960s [92]:

A *partially ordered set* is a pair $(P, \sqsubseteq)$, where P is a (possibly infinite) carrier set and $\sqsubseteq$ is a binary relation on P that is reflexive, transitive, and antisymmetric. A *complete lattice* is a partially ordered set $(L, \sqsubseteq)$ in which every subset $S \subseteq L$ has both a *supremum* (aka least upper bound) $\bigsqcup S \in L$ and an *infimum* (aka greatest lower bound) $\bigsqcap S \in L$. Every complete lattice has a *least* and a *greatest* element denoted by $\bot$ and $\top$, respectively. Given two complete lattices $(L, \sqsubseteq)$ and $(L', \sqsubseteq')$, a function $f\colon L \to L'$ is *monotonic* iff for any $x, y \in L$, $x \sqsubseteq y$ implies $f(x) \sqsubseteq f(y)$. Given a (monotonic) function f, a *prefixed point* is any x such that $f(x) \sqsubseteq x$; Analogously, a *postfixed point* is any x such that $x \sqsubseteq f(x)$. A *fixed point* is a point that is both a pre- and a postfixed point, i.e., $f(x) = x$.

The Knaster-Tarski theorem [63,97,70] states that every monotonic function (also referred to as *operator*) f admits a complete lattice of (possibly infinitely many) fixed points, where we denote the *least fixed point* (LFP) by $\mathsf{lfp} f$ and the *greatest fixed point* (GFP) by $\mathsf{gfp} f$. Here is a concise example adapted from [13]:

Example 1 (Complete Lattice of Fixed Points). Consider a carrier set $L = \{a, b, c\}$ ordered by $a \sqsubseteq b \sqsubseteq c$ and a monotonic operator f with $f(a) = a$, $f(b) =$

[3] We use the λ-expression $\lambda x\boldsymbol{.}\; M$ to denote an anonymous function which maps an element m to $M(x/m)$, i.e., the substitution of x by m in expression M.

$c, f(c) = c$. Then, we have $\mathsf{lfp} f = a$ and $\mathsf{gfp} f = c$, which are the only two fixed points of f that trivially form a complete lattice. ◁

Due to the mathematical duality between $\mathsf{lfp} f$ and $\mathsf{gfp} f$ [97], we focus on the least fixed point $\mathsf{lfp} f$ throughout the rest of the paper. The *fixed-point verification problems*, in general lattice-theoretic terms, can then be formulated as follows.

Given a complete lattice $(L, \sqsubseteq)$, a monotonic operator $f\colon L \to L$, and candidate lower bound $l \in L$, upper bound $u \in L$, LFP $I \in L$, determine

$$l \overset{\text{(i)}}{\sqsubseteq} \mathsf{lfp} f \overset{\text{(iii)}}{=} I \overset{\text{(ii)}}{\sqsubseteq} u \text{ , where} \tag{†}$$

(i) *Lower-bound verification*: prove that $l \sqsubseteq \mathsf{lfp} f$;
(ii) *Upper-bound verification*: prove that $\mathsf{lfp} f \sqsubseteq u$;
(iii) *LFP verification*: prove that $\mathsf{lfp} f = I$.

We remark that some techniques in the literature are proposed to address the *refutation* counterpart of the above verification problems, e.g., latticed bounded model checking [13] for refuting upper bounds on $\mathsf{lfp} f$, i.e., for certifying $\mathsf{lfp} f \not\sqsubseteq u$. In contrast to the verification perspective, the *fixed-point synthesis problems* aim to *generate* $l, u, I \in L$ such that Eq. (†) holds. In this paper, the term *fixed-point reasoning* refers to both problems of fixed-point verification and synthesis.

2.2 General Fixed-Point Reasoning Techniques

There exists in the literature a vast array of fixed-point theorems [70] pertaining to different branches of mathematics, e.g., domain theory, algebraic geometry, topology, and logics.[4] For the former, in particular, the Kleene fixed-point theorem [1,70] provides a constructive means of obtaining the LFP of continuous operators. An operator f is *continuous* (aka *Scott-continuous*) if, for any ascending chain $H = \{h_1 \sqsubseteq h_2 \sqsubseteq \cdots\}$ over the complete lattice $(L, \sqsubseteq)$, $f(\bigsqcup H) = \bigsqcup f(H)$, where $f(H)$ is the shorthand for $\{f(h) \mid h \in H\}$. Note that every continuous operator is monotonic [56, Theorem A.4]. The Kleene fixed-point theorem states that the LFP of any continuous operator f exists and can be constructed using *fixed-point iteration* from the least element $\bot \in L$:

$$\mathsf{lfp} f = \bigsqcup\nolimits_{n \in \mathbb{N}} f^n(\bot) = \lim\nolimits_{n \to \omega} f^n(\bot) \text{ .} \tag{‡}$$

This result has been further extended by Cousot and Cousot in [33] to monotonic (not necessarily continuous) operators via transfinite iterations.

The fixed-point iteration (‡) à la Kleene fixed-point theorem provides a simple mechanism to *verify lower bounds* or to *refute upper bounds* on least fixed points: One can certify (i) $l \sqsubseteq \mathsf{lfp} f$ by finding $k \in \mathbb{N}$ such that $l \sqsubseteq f^k(\bot)$; and (ii) $\mathsf{lfp} f \not\sqsubseteq u$

[4] We call these results *general* techniques as they are often expressed in an abstract, mathematical manner that is not tied to a concrete application in formal verification.

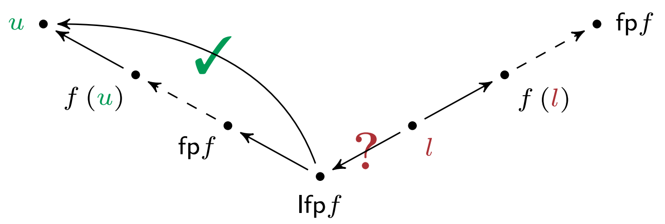


Fig. 2: Intuition of the soundness of Park induction (left branch) and the unsoundness of simple lower induction (right branch) [37]. An arrow from $h_1 \in L$ to $h_2 \in L$ indicates $h_1 \sqsubseteq h_2$. For Park induction, the iteration of f on u converges downwards to a fixed point of f which is – by the Knaster-Tarski theorem – necessarily above $\mathsf{lfp}f$. For simple lower induction, however, the ascending chain $l \sqsubseteq f(l) \sqsubseteq \cdots$ converges to a fixed point of f which is necessarily below $\mathsf{gfp}f$, but we do not know how l compares to $\mathsf{lfp}f$.

by finding $k \in \mathbb{N}$ such that $f^k(\bot) \not\sqsubseteq u$. The former constitutes the basis of *value iteration* [5,86,50] – a well-known technique for determining reachability probabilities in finite Markov models (see Section 5); Whilst the latter underpins *latticed bounded model checking* [13] – a latticed analogue of the well-established technique of bounded model checking [20,32,19].

However, reasoning about the LFP through fixed-point iteration as per Eq. (‡) is deemed inefficient in practice due to its iterative nature. In contrast, the Park induction principle [83] gives an elegant way of *verifying upper bounds* on LFPs:

$$f(u) \sqsubseteq u \quad \text{implies} \quad \mathsf{lfp}f \sqsubseteq u\ , \tag{1}$$

i.e., if pushing a candidate upper bound u through the (monotonic) operator f *once* yields a smaller element in terms of $\sqsubseteq$, then we have verified that u indeed upper-bounds $\mathsf{lfp}f$; see an illustration of the soundness of Park induction in (the left branch) Fig. 2. For instance, in Example 1, we can assert $\mathsf{lfp}f \sqsubseteq c$ since $f(c) \sqsubseteq c$. However, Park induction does not suffice to establish $\mathsf{lfp}f \sqsubseteq b$ (which is true as $\mathsf{lfp}f = a \sqsubseteq b$) because $f(b) = c \not\sqsubseteq b$. To remedy this, Batz et al. [13] proposed a strictly more general proof rule called *latticed k-induction* (or *κ-induction*, as an extension of k-induction [94]) stating that, for any $k \in \mathbb{N}$,

$$f\left(\Psi_u^k(u)\right) \sqsubseteq u \quad \text{implies} \quad \mathsf{lfp}f \sqsubseteq u\ , \tag{2}$$

where $\Psi_u \colon L \to L$ is the κ-induction operator that maps every element $h \in L$ to $f(h) \sqcap h$. The key idea behind Eq. (2) is to *rectify* $f(u)$ by "pulling it down" – via taking the greatest lower bound with u – in the hope that the rectified element $\Psi_u(u)$ satisfies Park induction and thus the candidate upper bound u can be verified. Albeit being strictly more general than Park induction, latticed k-induction remains incomplete:[5] Recall Example 1, the valid upper bound b can never be proved by κ-induction since, for any $k \in \mathbb{N}$, $\Psi_b^k(b) = b$ but $f(b) = c \not\sqsubseteq b$.

Next, we investigate induction rules for establishing lower bounds. It is not hard to show that a "dual" version of Park induction – by flipping $\sqsubseteq$ in Eq. (1)

[5] Completeness holds in the transfinite setting assuming unique fixed point [13].

– works for certifying lower bounds on the *greatest* fixed point $\mathsf{gfp} f$, but not on $\mathsf{lfp} f$. More concretely, for $l \in L$, the *simple lower induction* rule

$$l \sqsubseteq f(l) \quad \text{implies} \quad l \sqsubseteq \mathsf{lfp} f \ , \qquad ↯$$

is *unsound* in general; see an illustration in (the right branch of) Fig. 2.

To obtain sound proof rules for establishing lower bounds on LFPs, Hark et al. [47] proposed to retrieve the soundness of lower induction by adding *side conditions* therein capturing the notion of uniform integrability from stochastic processes. Baldan et al. [6] proposed alternative proof rules for specific lattices of the form $\mathbb{M}^Y$, where Y is a finite set and $\mathbb{M}$ is a special kind of lattice with an algebraic structure called MV-algebra [79]; The basic idea is as follows: For a non-expansive (and thus monotonic) function $f \colon \mathbb{M}^Y \to \mathbb{M}^Y$, each candidate lower bound l on $\mathsf{lfp} f$ is associated with a map $f_*^l \colon 2^{[Y]_{l=f(l)}} \to 2^{[Y]_{l=f(l)}}$, where $[Y]_{l=f(l)}$ denotes the set $\{y \in Y \mid l(y) \neq \bot \text{ and } l(y) = f(l)(y)\}$.[6] If l is a postfixed point, i.e., $l \sqsubseteq f(l)$, and $\mathsf{gfp} f_*^l = \emptyset$, then we have $l \sqsubseteq \mathsf{lfp} f$. This proof rule is a consequence of the following result on *verifying candidates for the exact LFP*:

Given a non-expansive function $f \colon \mathbb{M}^Y \to \mathbb{M}^Y$ and an LFP candidate $I \in \mathbb{M}^Y$ with $f(I) = I$, Baldan et al. (ibid.) associate I with a function $f_\#^I \colon 2^Y \to 2^Y$, whose greatest fixed point $\mathsf{gfp} f_\#^I$ intuitively encodes the *potential* for increasing I by adding a constant. Thus, if $f(I) = I$ and $\mathsf{gfp} f_\#^I = \emptyset$, then $\mathsf{lfp} f = I$. This approach transforms the potentially infinite domain $\mathbb{M}^Y$ to the finite domain 2^Y, thereby facilitating the verification of the LFP candidate. Furthermore, given that I is a fixed point, the implication is *reversible*: $\mathsf{gfp} f_\#^I = \emptyset$ is also a necessary condition for $\mathsf{lfp} f = I$.

Apart from verifying candidates for exact LFPs, Baldan et al. [7] also proposed a game-theoretic approach to *computing the exact LFP* by solving a simple parity game called *fixpoint game*, which provides a complete characterization of the solutions of fixed-point equations over continuous lattices [91]. Such simple parity games can be solved by, e.g., Jurdziński's algorithm [55] together with Calude et al.'s improvement [23].

Fixed points in metric spaces. Most fixed-point characterizations for discounted properties of Markov decision processes (see Section 5) root in the Banach fixed-point theorem (aka the contraction mapping theorem) [8] in the theory of metric spaces [2]. A map $f \colon X \to X$ over a metric space (X, d) with metric d is called a *contraction mapping* on X if there exists $\epsilon \in [0, 1)$ such that $\forall x_1, x_2 \in X \colon d(f(x_1), f(x_2)) \leq \epsilon d(x_1, x_2)$. The Banach fixed-point theorem states that any contraction mapping f on a (non-empty) complete metric space [2] admits a *unique* fixed point $\mathsf{fp} f$; Moreover, starting from an *arbitrary* element $x_0 \in X$, the sequence $\{f^n(x_0)\}_{n \in \mathbb{N}}$ necessarily converges to the unique fixed point $\mathsf{fp} f$.

Mardare et al. [74] extended the quantitative equational logic [73] for metric spaces by incorporating fixed-point operators and approximate axioms. They

[6] Consider $l \in \mathbb{M}^Y$ as a map $Y \to \mathbb{M}$.

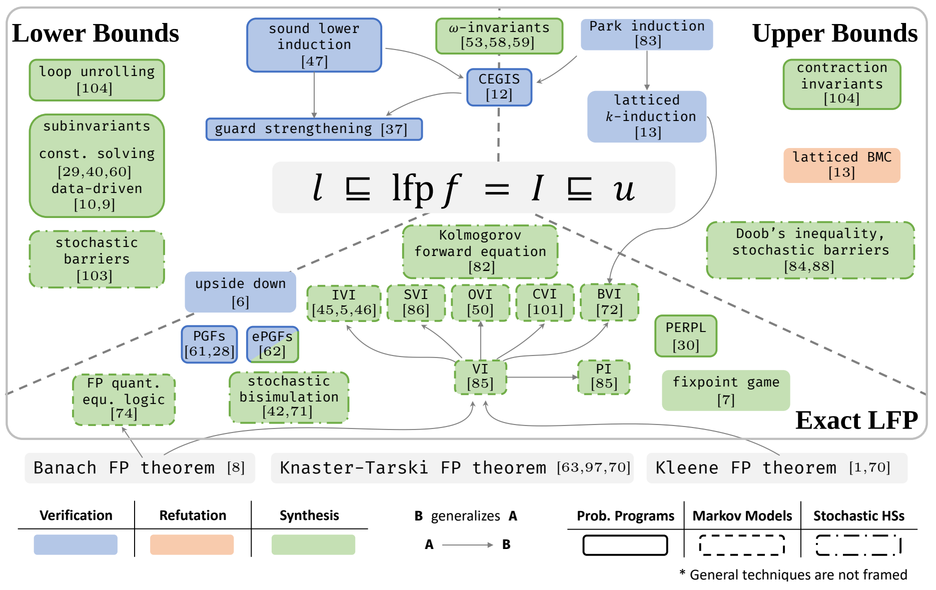


Fig. 3: The quantitative fixed-point reasoning landscape for stochastic systems.

introduced the concept of Banach pattern and established proof rules to capture contractiveness of functions and to reason about fixed points based on the Banach fixed-point theorem.

3 The Quantitative Fixed-Point Reasoning Landscape

This section sketches our (again, far-from-exhaustive) *fixed-point reasoning landscape for stochastic systems* as depicted in Fig. 3, where we focus on core techniques for *verifying*, *refuting*, or *synthesizing* upper bounds (over-approximations), lower bounds (under-approximations), or exact values for the least fixed point $\mathsf{lfp} f$ of a monotonic operator f (cf. Eq. (†)). More details of these techniques as well as a set of closely related works will be examined in subsequent sections.

The underlying stochastic models we consider include probabilistic programs [64,44,76,11], probabilistic transition systems (more specifically, Markov models [4,35]), and stochastic hybrid systems [51,24]. In particular, we emphasize the *quantitative* nature of fixed-point reasoning techniques for these systems since – in contrast to *qualitative* properties such as (Boolean) reachability and safety – we are often concerned with *quantities* like assertion-violation probabilities [100], expectations [75], moments [69,99,78], expected runtimes [59], and concentrations [25,27]. We will showcase the fixed-point encodings for a prominent subset of these quantities in Sections 4 to 7.

Remark 2. Granted that the above-mentioned fields have been extensively studied for decades and numerous results are established thereof in terms of verification, refutation, or synthesis. However, we include in Fig. 3 only a selected subset of related works where the problems are *explicitly* formulated using LFPs. $\triangleleft$

4 Fixed-Point Reasoning for Probabilistic Programs

Probabilistic programming is a widely used paradigm to describe stochastic systems in the form of executable computer programs. It extends classical, deterministic programming with abilities of *random sampling* and *conditioning* (via posterior observations). The so-obtained probabilistic programs [64,44,76,11] are typically normal-looking programs describing *posterior probability distributions*. Reasoning about possibly unbounded loops – whose (denotational) semantics is often characterized as the least fixed point of a certain monotonic operator – is one of the most difficult tasks in the quantitative verification of probabilistic programs. This section surveys recent advancements in this line of research using a commonly studied probabilistic programming language called the *probabilistic guarded command language* (pGCL) [75], which augments Dijkstra's GCL [36] with nondeterminism and randomness.

4.1 The Probabilistic Guarded Command Language

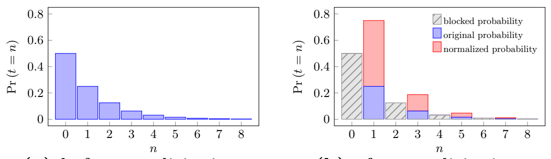
A pGCL program C, in general, adheres to the following grammar:

$$
\begin{array}{rcll}
C & ::= & \texttt{skip} & \text{(effectless program)} \\
 & \mid & \texttt{diverge} & \text{(freeze)} \\
 & \mid & x := E & \text{(assignment)} \\
 & \mid & x :\approx \mu & \text{(random assignment)} \\
 & \mid & C \fatsemi C & \text{(sequential composition)} \\
 & \mid & \texttt{if}\ (\varphi)\{C\}\ \texttt{else}\ \{C\} & \text{(conditional choice)} \\
 & \mid & \{C\} \ \Box\ \{C\} & \text{(nondeterministic choice)} \\
 & \mid & \{C\}\ [p]\ \{C\} & \text{(probabilistic choice)} \\
 & \mid & \texttt{while}\ (\varphi)\{C\} & \text{(while loop)} \\
 & \mid & \texttt{observe}\ (\varphi) & \text{(conditioning)}
\end{array}
$$

Here, x is a variable from a finite set Vars, E denotes an arithmetic expression, μ is a probability distribution, and φ is a Boolean condition defined over program variables. `skip` does nothing and simply continues the execution, while `diverge` halts execution indefinitely. Assignments, such as $x := E$, update a variable x with the result of E, and random assignments $x :\approx \mu$ assign a value to x sampled from the distribution μ. Sequential composition $C_1 \fatsemi C_2$ executes C_1 followed by C_2. Conditional choice `if` $(\varphi)\{C_1\}$ `else` $\{C_2\}$ selects C_1 if φ evaluates to true and C_2 otherwise, while nondeterministic choice $\{C_1\}\ \Box\ \{C_2\}$ nondeterministically selects between C_1 and C_2. Probabilistic choice $\{C_1\}\ [p]\ \{C_2\}$ executes C_1 with probability p and C_2 with probability $1 - p$. Loop `while` $(\varphi)\{C\}$ repeatedly execute C as long as φ evaluates to *true*. Finally, `observe` (φ) statement, restricts execution to traces where φ holds.

```
$h \coloneqq 1\,⨾$
while $(\,h = 1\,)\,\{$
    $\{\,t \coloneqq t+1\,\}\,[\,{}^1\!/\!_2\,]\,\{\,h \coloneqq 0\,\}$
$\}\,⨾$
observe $(\,t \equiv 1 \pmod 2\,)$
```

Prog. 1: Odd geometric dist.



(a) before conditioning **(b)** after conditioning

Fig. 4: Snippets of the distribution of t in Prog. 1.

Example 2 (Odd Geometric Distribution [62]). Prog. 1 illustrates an iterative algorithm that flips a fair coin while counting the number of trials (t) until a tail ($h = 0$) is observed, and then conditions on t being odd. The conditioning-free `while` loop

$$\mathit{Loop}_{\text{geo}}: \quad \texttt{while}\ (h = 1)\{\, t \coloneqq t + 1\ [{}^1\!/\!_2]\ h \coloneqq 0\,\}$$

generates a geometric distribution over variable t and variable h tracks the presence of a head ($h = 1$) or a tail ($h = 0$) in each trial. Following the $\mathit{Loop}_{\text{geo}}$, the statement `observe`$(t \bmod 2 = 1)$ filters out all runs where t is even, ensuring that only odd values of t are considered. The interaction between the probabilistic loop and post-execution conditioning is shown in Fig. 4.

4.2 Fixed-Point Semantics for Probabilistic Loops

The (denotational) semantics of a probabilistic loop is often characterized as the least fixed point of a monotonic operator over a complete lattice structure. We next present three well-known fixed-point encodings leveraging the weakest preexpectation calculus and the expected runtime calculus (both are known as backward semantics), as well as the semantics based on probability generating functions (a typical forward semantics).

Weakest preexpectation. For a probabilistic program, a program state maps each variable in Vars to $\mathbb{Q}$. The set of all program states can thus be represented as $\Sigma = \{\sigma \colon \mathsf{Vars} \to \mathbb{Q}\}$. An expectation [56,75] is a mapping from program states to $\overline{\mathbb{R}}_{\geq 0}$, where $\overline{\mathbb{R}}_{\geq 0} = \{r \in \mathbb{R} \mid r \geq 0\} \cup \{\infty\}$, and the set of all expectations is denoted as $\mathbb{E} = \left\{e \mid e \colon \Sigma \to \overline{\mathbb{R}}_{\geq 0}\right\}$. There exists a complete lattice $(\mathbb{E}, \sqsubseteq)$, where the partial order $\sqsubseteq$ is defined pointwise: $\forall e_1, e_2 \in \mathbb{E}$, $e_1 \sqsubseteq e_2$ iff $\forall \sigma \in \Sigma \colon e_1(\sigma) \leq e_2(\sigma)$. The *weakest preexpectation transformer* [75] $\mathrm{wp}[\![\cdot]\!] \colon \mathrm{Progs} \to (\mathbb{E} \to \mathbb{E})$ provides a formal method for reasoning about the expected outcomes of probabilistic programs. Let $[\varphi]$ denote the casting of the Boolean expression φ into an expectation. For a loop construct `while` $(\varphi)\{C\}$ and a postexpectation g, the weakest preexpectation (WP) $\mathrm{wp}[\![\texttt{while}\ (\varphi)\{C\}]\!](g)$ is the least fixed point of the monotonic and continuous mapping

$$\lambda X.\ [\neg\varphi] \cdot g + [\varphi] \cdot \mathrm{wp}[\![C]\!](X)$$

on the complete lattice $(\mathbb{E}, \sqsubseteq)$. For instance, In Example 2, weakest preexpectation of $\mathit{Loop}_{\text{geo}}$ w.r.t. postexpectation $t+h$ is the least fixed point of the mapping $\lambda X\text{.}\ [h \neq 1] \cdot (t+h) + [h=1] \cdot ({}^1\!/_2 \cdot X\,(t/t+1) + {}^1\!/_2 \cdot X\,(h/0))$, where $X\,(x/e)$ represents the substitution of variable x by expression e in expectation X.

Expected runtime. Kaminski et al. [58,59] proposed a wp-style calculus for analyzing expected runtimes (ERT) of probabilistic programs. For a probabilistic program C, the runtime maps each state σ to the average or expected runtime of C. The space of all runtimes is defined as $\mathbb{T} = \{t \mid t\colon \Sigma \to \overline{\mathbb{R}}_{\geq 0}\}$. Similar to $(\mathbb{E}, \sqsubseteq)$, the structure $(\mathbb{T}, \sqsubseteq)$ also forms a complete lattice under a pointwise partial order. The *expected runtime transformer*, $\mathrm{ert}[\![\cdot]\!]\colon \mathrm{Progs} \to (\mathbb{T} \to \mathbb{T})$, is defined analogously to the weakest preexpectation transformer. For a loop construct `while` $(\varphi)\{C\}$, assuming the runtime of the program segment following this loop is f, the expected runtime of `while` $(\varphi)\{C\}$ up to the end of the program is the least fixed point of the monotonic and continuous mapping

$$\lambda X\text{.}\ 1 + [\neg\varphi] \cdot f + [\varphi] \cdot \mathrm{ert}[C](X)$$

on the complete lattice $(\mathbb{T}, \sqsubseteq)$. In Example 2, the expected runtime of $\mathit{Loop}_{\text{geo}}$ is the least fixed point of the mapping $\lambda X\text{.}\ 1 + [h=1] \cdot (1 + {}^1\!/_2 \cdot X\,(h/0) + {}^1\!/_2 \cdot X\,(t/t+1))$.

In addition to expected runtime, there is also wp-style fixed-point encoding for *resource analysis* of probabilistic programs [81], which is not elaborated here.

Probability generating functions. The representation of program variable distributions can be expressed using probability generating functions (PGFs) [61,28], probability density functions (PDFs) [43,52,80,17,18], or measures [64,34,104]. Here, we take PGFs as an example. A PGF encodes a possibly infinite-support discrete distribution in the form of a formal power series:

$$F \ = \ \sum\nolimits_{n \in \mathbb{N}} a_n X^n \ ,$$

where X is a formal indeterminate corresponding to program variable x and a_n stands for the probability of $x = n$. For instance, the geometric distribution of variable t in Example 2 can be encoded as the following PGF:

$$\frac{1}{2} + \frac{1}{4}T + \frac{1}{8}T^2 + \frac{1}{16}T^3 + \frac{1}{32}T^4 + \frac{1}{64}T^5 + \frac{1}{128}T^6 + \cdots \ .$$

Moreover, PGFs can compress infinite series into a closed form using Taylor's theorem: the above PGF can be represented in the closed form $\frac{1}{2-T}$. This PGF is in fact the least fixed point of the characteristic function of $\mathit{Loop}_{\text{geo}}$ (with initial values $h = 1, t = 0$) over a complete lattice of PGFs. The PGF technique has been extended in [62] to cope with Bayesian inference with conditioning.

4.3 Reasoning about Fixed-Point Semantics

In this section, we introduce specific fixed-point reasoning techniques in the semantics of probabilistic programs. To facilitate a unified discussion, we denote the complete lattice and monotonic and continuous mappings from Section 4.2 as $(\mathbb{L}, \sqsubseteq)$ and Φ, respectively. Furthermore, we assume that Φ is a mapping on $(\mathbb{L}, \sqsubseteq)$.

Upper-bound Reasoning. Park Induction [83] provides a simple yet effective sufficient condition for verifying upper bounds on least fixed points. The element $I_{sup} \in \mathbb{L}$ that satisfies the conditions of Park Induction is referred to as *superinvariant*. Building on Park Induction, techniques for verifying upper bounds have been further developed by synthesizing $I_{sup} \sqsubseteq f$, where f represents the upper bound being verified. Notable contributions in this direction include the instantiation of latticed k-induction for probabilistic programs [13]. Subsequently, Batz et al. [12] proposed an efficient framework for inductive synthesis that operates directly at the source-code level. This framework leverages a template-based strategy in conjunction with a counterexample-guided inductive synthesis (CEGIS) loop to automate the construction of I_{sup}. The continuity of the operator Φ greatly facilitates upper bound refutation. Utilizing this property, Batz et al. [13] introduced latticed bounded model checking and instantiated it in tasks related to probabilistic programs. Representative works in upper bound synthesis include upper ω-invariants [58,59] based on sequence construction and contraction invariants [104] via distribution rearrangement.

Lower-bound Reasoning. Lower bounds synthesis is relatively straightforward, as $\perp$ is a natural lower bound that serves as a starting point for fixed-point iteration. Due to the continuity of the operator Φ, this iterative computation can generate lower bounds with arbitrary precision. This principle has been applied through loop unrolling [104] to generate lower bounds for posterior distributions. Another prominent approach for constructing lower bounds involves the synthesis of lower ω-invariants [53,58,59], which relies on constructing sequences and computing their limits. Verification of lower bounds was first systematically addressed by McIver and Morgan's induction [75], which, however, is limited to bounded expectations and requires prior knowledge of the termination probability of the `while` loops. Although lower bound verification may appear straightforward – for instance, by generating a high-precision lower bound f' such that $f \sqsubseteq f'$ to verify $f \sqsubseteq \text{lfp}\ \Phi$ – this process often involves complex iterative computations, making it inefficient and impractical in many cases. In fact, efficiently verifying lower bounds is significantly more challenging than verifying upper bounds, primarily due to the lack of a simple rule analogous to Park Induction. Specifically, the rule $I \sqsubseteq \Phi(I) \implies I \sqsubseteq \text{lfp}\ \Phi$ is generally unsound for lower bounds verification [47]. The element $I_{sub} \in \mathbb{L}$ that satisfies $I_{sub} \sqsubseteq \Phi(I_{sub})$ is called *subinvariant*. Typical techniques on the synthesis of I_{sub} include data-driven approaches [10,9] and constraint solving [29,40,60]. The unsoundness of I_{sup} for lower bounds verification in weakest preexpectation reasoning was addressed by Hark et al. [47] through the concept of *uniform integrability*. They also proposed an inductive proof rule for verifying lower bounds on WP. Building on this work, Batz et al. [12] generalized their CEGIS framework to automate lower bounds verification by proving UPAST properties and synthesizing c.d.b. subinvariants.

Notably, many lower bounds verification approaches, including [47] and [12], are limited to probabilistic programs that are almost-surely terminating. To address this limitation, Feng et al. [37] proposed a proof rule leveraging the technique of guard strengthening, thereby unleashing the general applicability of these approaches to possibly divergent probabilistic programs.

Exact inference. PERPL [30] is a recently proposed probabilistic programming language that models unbounded recursion using systems of polynomial equations. Its semantics are defined based on Markov kernels, and it leverages numerical methods to compute least fixed points directly from the compiled equations. Chen et al. [28] introduced a decidability result for determining whether such programs can generate a specified distribution, leveraging probability generating functions to represent and manipulate both finite and infinite-support distributions efficiently. Klinkenberg et al. [62] introduce an exact Bayesian inference method for probabilistic programs with unbounded loops, using probability generating functions to handle the intricacy of conditioning.

5 Fixed-Point Reasoning for Markov Models

Markov models are mathematical frameworks used to describe systems with transitions between states in a probabilistic manner. They are extensively employed in various fields, including chemistry, biology, machine learning, and physics [85,77]. In practical applications, Markov models often involve a large number of states and complex structures, such as cycles (or, loops). To analyze these systems, researchers commonly formulate a wide range of problems as the computation of a fixed point of a specific function. This section delves into the theoretical foundations of fixed-point reasoning in Markov models. By examining fundamental problems and approaches to identifying fixed points, we aim to provide a rigorous yet accessible perspective on this essential topic.

5.1 Introduction to Markov Models

A Markov chain (MC) is a probabilistic transition model in which the transition probability depends solely on the current state, with no influence from previous states. Commonly studied problems in Markov Chains include calculating reachability probabilities and expected rewards. Formally, an MC is a pair (S, δ), where S is a set of states and $\delta \colon S \to \mathit{Dist}(S)$ is a function mapping each state to a distribution of successor states.

A Markov decision process (MDP) extends the concept of MC by introducing the notion of "decision". In an MDP, transitions to the next state are influenced by *policies*, each of which represents a specific distribution of transition probabilities. Formally, an MDP is a tuple $(S, \mathit{Act}, \delta)$, where S is a set of states, Act is a set of actions, and $\delta \colon S \times \mathit{Act} \to \mathit{Dist}(S)$ is a function mapping a pair of a state together with an action to the distribution of successor states. A policy is a function $S \to \mathit{Act}$, mapping each state to a certain action. With a policy, an

MDP degenerates into an MC. *Decision-making* in MDPs refers to the problem of finding a "good" policy that optimizes certain objectives.

Since an MC can always be considered as a special MDP with a singleton Act, we focus our discussion on MDPs without loss of generality.

A *weighted* MDP is a tuple (S, Act, δ, T, R), where (S, Act, δ) is an MDP, $T \subseteq S$ is a set of target states, and $R\colon S \times Act \times S \to \mathbb{R}$ is a weight function (sometimes also called reward).[7] For a weighted MDP, we fix a policy π. A path is a sequence of states $(s_i)_{i\geq 1}$ with each transition from s_i to s_{i+1} being $\delta(s_i, \pi(s_i))(s_{i+1})$. In a weighted MDP with a set of target states T, a path either ends in a state $s \in T$ or is infinite and contains no states in T. Let P_π denote the set of all paths and μ_π denote the probabilistic measure over P_π.

5.2 Discounted and Undiscounted Objectives

Given a weighted MDP (S, Act, δ, T, R) and a fixed policy π, the *accumulated weight (or, reward)* of a path $p = (s_i)_{i\geq 1}$ is

$$\sum\nolimits_{i\in\mathbb{N}} R\left(s_i, \pi(s_i), s_{i+1}\right) \cdot \gamma^i \ ,$$

where $\gamma \in (0, 1]$ is a discount factor. We can formulate almost all the problems related to MCs/MDPs to a general form, that is, to evaluate the expected accumulated weight of paths over μ_π and find the best policy that maximizes or minimizes the expected accumulated weight.

According to different configurations of R and γ, the objectives concerned with MCs and MDPs can be generally divided into the following 3 classes [16]:

- *Discounted reward objectives.* The discount factor γ is strictly smaller than 1. Discounted reward objective has a wide range of applications. It serves as the core of Reinforcement Learning [96].
- *Weighted reachability objectives.* The discount factor $\gamma = 1$, $R(s, a, S') \geq 0$ for all $s, s' \in S$ and $a \in Act$, $R(s, a, s') = 0$ for $s' \notin T$, and $R(s, a, s')$ is a constant for a fixed s'. Intuitively speaking, if a path ends in a target state s, it is assigned a reward that is only related to s. Otherwise, the reward is 0. Solving weighted reachability objectives means optimizing the expected reward. If $R(s, a, s') = 1$ for all $s' \in S$, this degenerates to optimizing the probability of reaching T.
- *Stochastic shortest path objectives.* The discount factor $\gamma = 1$ and $R(s, a, s') > 0$ for all $s, s' \in S$ and $a \in Act$. Intuitively speaking, if a path ends in a target state s, its (weighted) length is the accumulated weight of all the transitions along the path. Otherwise, the length is infinity. Solving stochastic shortest path means optimizing the expected path length.

Optimizing expected accumulated weight can be formulated as the computation of a fixed point for a specific function called the *Bellman update operator*:

$$\mathit{BellmanUpdate}(V) \ = \ V' \ , \tag{3}$$

[7] A generalization to $\mathbb{R} \cup \{\infty\}$ with *infinite rewards* is recently proposed in [14].

where V and V' are *value vectors* of the form $S \to \mathbb{R}$ and V' is[8]

$$V'(s) \;=\; \sup_{a \in Act} \mathbb{E}_{s' \in S}(\gamma V(s') + R(s,a,s'))\ . \tag{4}$$

The fixed-point equation $BellmanUpdate(V) = V$ is called the *Bellman equation*. For discounted reward objectives, where $\gamma < 1$, $BellmanUpdate$ satisfies the definition of a contraction mapping. This enables the application of the Banach fixed-point theorem, allowing the iterative computation of a unique fixed point starting from any initial value. However, this method does not extend to undiscounted objectives, which are of primary interest in the context of verification. In such cases, a (complete) lattice structure over $[0,\infty]^{|S|}$,[9] where $|S|$ represents the size of the state space, is typically employed to address the problem. $BellmanUpdate$ is a monotone function under the natural partial order defined on the lattice, making it amenable to analysis via Kleene's fixed-point theorem. Starting from the least element $\bot$, the iterative application of $BellmanUpdate$ yields the least fixed point, which often aligns with the desired solution. In both discounted and undiscounted cases, this iterative process for computing the fixed point is known as *value iteration* (VI). For additional details, refer to [4].

5.3 Reasoning about the Bellman Equation in Markov Models

As mentioned earlier, the most straightforward way for computing fixed points of Bellman update operators is VI. There are other mainstream approaches based on *linear programming* (LP) and *policy iteration* (PI) [85]:

- PI is also an iterative approach that iterates on policies instead of value vectors. It alternates between two key steps: policy evaluation and policy improvement. In policy evaluation, a policy is fixed so that the MDP degenerates into an MC. Then we evaluate the fixed point of the MC. In policy improvement, a new policy is derived using the value of the fixed point in the policy evaluation step. This process repeats until the policy stabilizes, indicating optimality.
- The computation of the expected accumulated weight can also be formulated as a linear programming problem. This formulation allows for a polynomial-time complexity in the number of states $|S|$, whereas both VI and PI have exponential worst-case time complexity. However, both VI and PI require only linear space, while a linear-space algorithm for LP has not yet been discovered.

Hartmans et al. [49] provided a thorough experimental overview of VI, PI, and LP in the context of probabilistic model checking. They compared different state-of-the-art algorithms and LP solvers in terms of performance and correctness.

Since VI has a more direct relationship with fixed-point reasoning, the subsequent discussion primarily focuses on VI and its variants.

[8] Here, sup corresponds to maximizing the expected accumulated weight. For the minimization counterpart, use inf instead.

[9] Sometimes $[0,1]$ or other forms. This depends on the definition of R and the specific objective.

Establishing stopping criteria for value iteration. The VI approaches, in their early form, hardly admit rigorous stopping criteria. A classical and naive approach is that, for the current value V and a given error bound ϵ, the VI algorithm terminates when $|BellmanUpdate(V) - V| < \epsilon$ or $|BellmanUpdate(V) - V|/|V| < \epsilon$. These stopping criteria perform well in a number of cases and are implemented in dominant probabilistic model checkers. However, there is no guarantee for the final result of the naive approach. Haddad et al. [45] studied this problem and first pointed out that the algorithm can return results with arbitrary errors. They presented an *interval value iteration* (IVI) algorithm together with a straightforward stopping criterion. Meanwhile, Brázdil et al. [21] introduced a verification method for MDPs based on machine learning together with stopping criteria for the VI algorithm to avoid excessive useless iterations. Křetínský et al. [66] provided the first stopping criterion for VI of stochastic games with total rewards and mean payoffs. They presented corresponding solutions through two approaches: reduction to MDPs and direct operation on stochastic games and illustrated that the latter approach is more efficient.

Optimizations on value iteration. Baier et al. [5] proposed an interval iteration method, improving the accuracy of computing expected cumulative costs. Their method can simultaneously provide upper and lower bounds for expected cumulative costs and ensures convergence of iterations on both ends. Mathur et al. [15] presented a new algorithm that improves the accuracy of approximate results for iterative algorithms like VI. They implemented an extension of the PRISM model checker [67] for this algorithm, enhancing the practicality of probabilistic model checkers. Quatmann et al. [86] considered the selection of initial values in the value iteration algorithm and proposed *sound value iteration* (SVI), which does not require priori computations of the initial vectors. SVI exhibits faster convergence and can yield tight upper and lower bounds; its applicability has been demonstrated through the calculation of reachability probability and can be easily extended to expected rewards. Hartmanns et al. [50] proposed *optimistic value iteration* (OVI) which first computes a lower bound through standard VI, then "guesses" an upper bound based on this lower bound, and finally verifies the correctness of the upper bound. OVI is integrated into the probabilistic model checking toolchain MODEST TOOLSET [48]. For systems admitting a unique fixed point, Lu et al. [72] proposed *bisection value iteration* (BVI); it iteratively refines the result of a standard VI procedure while applying Park induction [83] and κ-induction [13] to check whether the intermediate result is a genuine lower/upper bound. Watanabe et al. [101] proposed *compositional value iteration* (CVI) leveraging a divide-and-conquer paradigm; it works on components of MDPs instead of the monolithic system and reuses the local, intermediate results to improve the scalability and efficiency of value iteration.

6 Fixed-Point Reasoning for Stochastic Hybrid Systems

Stochastic hybrid systems (SHSs) [51,24] are the stochastic counterparts of deterministic hybrid systems [98,31,41] modeling both continuous and discrete dynamic behaviors. They are widely used to represent stochastic behaviors in fields such as biology, control systems, and others. In SHSs, the discrete components are typically modeled using Markov models (cf. Section 5), while the continuous components are often represented by stochastic differential equations (SDEs). In this section, we mainly focus on the continuous part of the SHSs.

A stochastic differential equation [82] takes the form

$$\mathrm{d}X_t \;=\; b(X_t)\;\mathrm{d}t + \sigma(X_t)\;\mathrm{d}W_t, \quad t \geq 0\;, \tag{5}$$

where $\{X_t\}$ is a continuous-time stochastic process, and $\{W_t\}$ is a Wiener process (standard Brownian motion). Here, b is the drift coefficient, governing the deterministic evolution of the system, and σ is the diffusion coefficient, capturing the system's coupling to the Gaussian white noise increments $\mathrm{d}W_t$. Unlike ordinary differential equations, where solutions are single-valued functions, solutions to stochastic differential equations (SDEs) are continuous-time stochastic processes. Let $F(t, X)$ denote the solution of Eq. (5) at time t with the initial random state X. The process X_t satisfies $F(t, X_h) = X_{t+h}$ thereby allowing X_t to be interpreted as a fixed point of F. Due to the functional nature of F, explicitly formulating the solution process $\{X_t\}_{t\geq 0}$ is equivalent to solving a partial differential equation, specifically the Kolmogorov forward equation [82], which is generally difficult to address. Consequently, research efforts have focused on characterizing the quantitative properties of stochastic hybrid systems (SHS).

Stochastic reachability [22], which quantifies the probability that a system eventually enters a target set, is a fundamental property examined in the verification of stochastic systems. In the seminal work by Prajna et al. [84], the concept of a stochastic barrier function was introduced to obtain upper bounds on stochastic reachability probabilities. The synthesis of a stochastic barrier function is essentially equivalent to identifying a prefixed point $B(x)$ satisfying

$$\mathrm{E}[\,B(X_{t_2}) \mid X_{t_1}\,] \;\leq\; X_{t_1}\;,$$

where X_t denotes the stochastic trajectory of a SHS. Since then, numerous studies have been dedicated to developing the *barrier-based* approach to obtain tighter upper bounds. For instance, [95] introduces c-martingale conditions for finite-time safety verification, while [89,88] extend barrier conditions to state-dependent constraints for finite-time safety. Additionally, [38] proposes vector exponential barriers, which achieve exponentially decreasing bounds on tail reachability probabilities, to name a few. Beyond reachability, barrier-based methods can also address other temporal properties such as reach-avoid, persistence, and recurrence. For example, [103] demonstrates how to upper- and lower-bound stochastic reach-avoid probabilities by leveraging the "ergodic occupation measure", and [26] introduces additive and multiplicative barrier rules for verifying persistence and recurrence properties.

In addition to barrier-based methods, several efforts attempt to approximate the exact fixed-point numerically based on discretization abstraction and probabilistic (bi-)simulation [68,3]. These methods typically begin by discretizing the state space to construct an abstract, more tractable system (e.g., finite-state), and then establish bounds on the differences between the original and abstract systems. In [42], Fränzle et al. give a high-level specification model of stochastic hybrid automata and demonstrate how to bisimulate and over-approximate a stochastic hybrid automaton by a probabilistic hybrid automaton. [54] proposes a stochastic bisimulation function that quantifies the distance between two linear SHS. [71] survey different types of closeness guarantees for both discrete and continuous stochastic systems.

7 Unified Frameworks for Fixed-Point Reasoning

As discussed in the previous sections, methods for approximating fixed points vary across different types of systems. Despite the diversity of approaches to the fixed-point reasoning problem in stochastic systems, several efforts have sought to provide a unified perspective on this issue.

Interperting Probabilistic Programs as Markov Models [39]. The semantics of a probabilistic program can be operationally interpreted as a Markov process defined over its underlying control flow graph. Specifically, for any given program C, there exists an associated Markov chain $\mathcal{M}_C = (S_C, \mathbf{P}_C)$, where S_C is defined as $L \times \mathsf{Val}$, representing the product of program counters and variable valuations (cf. Fig. 5). Consequently, fixed-point reasoning in probabilistic programs can be framed within the context of fixed-point reasoning in Markov models. Specifically, given a Markov chain $\mathcal{M} = (S, \mathbf{P})$ derived from a probabilistic program. For any subset $H \subseteq S$, functions $f \colon S \to \mathbb{R}$ and $g \colon S \to \mathbb{R}$, [39] introduces a novel weighted reachability value function $V(s)$ defined as follows:

$$V(s) \;=\; E_s\left[\sum_{i=0}^{T_H-1} f(X_i) + \mathbf{1}_{T_H<\infty} \cdot g(X_{T_H})\right] . \tag{6}$$

The key insight is that by appropriately selecting the functions f, g, and the set H, the value function $V(\cdot)$ can accurately represent various quantities such as the wp-/ert-transformer and other related metrics. This formulation effectively transforms the quantitative verification problem into an MDP problem with a weighted reachability objective. In the context of Markov models, the value function $V(s)$ satisfies the Bellman equation:

$$\begin{aligned} \int_S V(t)\,\mathbf{P}(s, \mathrm{d}t) + f(s) \;&=\; V(s) \quad , \text{ if } s \in S \backslash H \;, \\ V(s) \;&=\; g(s) \quad , \text{ if } s \in H \;. \end{aligned} \tag{7}$$

Moreover, the weakest preexpectation transformer rule directly follows from the Bellman equation, providing a link between fixed-point reasoning in probabilistic programs and Markov models.

```
1 :  while ($x \geq 0$){
2 :    if ($x \geq 4$)
3 :      {$x = x - 1$ [0.4] $x = x + 1$}
4 :    else
5 :      {$x = x - 1$ [0.5] $x = x + 1$}
6 :  }.
7 :
```

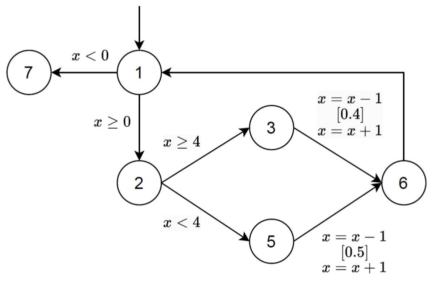


Fig. 5: A probabilistic program (left) and its CFG (right), where location label $l_{\mathsf{in}} = 1$ and $l_{\mathsf{end}} = 7$ represent the starting and ending point of the program, respectively.

Interpreting Markov models as abstract dynamic programming. As shown in Section 5, contraction mapping is a crucial property in fixed-point reasoning of Markov models. In [16], Bertsekas observes that monotonicity and contraction are two fundamental properties capturing the essence of dynamic programming.

In the framework of abstract dynamic programming, the set of value functions $J\colon X \to \mathbb{R}$ is denoted by $R(X)$, where X represents the abstract state space. The Bellman operator is defined as $H\colon X \times U \times R(X) \to R(X)$. Given a policy μ, the contraction map T_μ is formulated by

$$T_\mu J(x) \;=\; H(x, \mu(x), J)\;. \tag{8}$$

Therefore, the methods in abstract dynamic programming that compute or approximate the least fixed point (LFP) of $T_\mu J$ effectively encapsulate results pertinent to the verification of Markov models.

8 Challenges and Future Directions

This section briefly outlines several key challenges and potential research directions in the quantitative fixed-point reasoning for stochastic systems.

- *Exact inference of LFPs for probabilistic programs*: Albeit with a handful of preliminary results in this line of research, the limit of exact LFP inference remains obscure: On the one hand, the so-far identified subclasses of probabilistic programs admitting effective symbolic inference algorithms are limited in various ways, e.g., bounded loops, finite-support distributions, linear or even rectangular expressions, or the requirement of delicate invariants; On the other hand, can we identify "the most expressive fragment" beyond which exact inference is no longer possible? Does such fragment even exist?
- *Template-free approximations of LFPs*: While template-based methods for approximating LFPs are hindered by the difficulty in selecting suitable templates, only a few approaches have successfully pursued template-free strategies for LFP approximations. Notably, recurrence-based methods have been applied to probabilistic programs, and there have been few attempts using

numerical and statistical methods for stochastic systems. However, these existing techniques often fall short when being extended to general fixed-point reasoning due to their domain-specific constraints. A potential approach to advancing template-free LFP approximations lies in the appropriate combination of symbolic and numerical methods.

- *Automated synthesis of stochastic systems*: Existing synthesis techniques in the realm of stochastic systems, including those surveyed in this paper, are mostly concerned with quantitative expressions, e.g., invariants, expectations, moments, termination witnesses, probabilities, and policies. Extending the synthesis scope to expressive models of stochastic systems is an interesting direction. The main challenges are twofold: (i) identify stochastic dependencies between distributions through, e.g., sampled data; and (ii) identify system sketches with potentially complex iterative or cyclic behaviors. The tool of fixed-point reasoning may be leveraged to address these challenges.
- *Unified frameworks for FP reasoning*: Fixed-point reasoning exhibits consistent patterns across diverse models, including probabilistic programs, MDPs, and stochastic hybrid systems. Despite initial efforts to develop unified frameworks for fixed-point reasoning, identifying the common foundational elements that apply across these different application domains remains a significant challenge. Establishing such a unified approach is essential for advancing the generalizability and scalability of fixed-point reasoning methodologies in various stochastic and probabilistic contexts.

9 Conclusion

We have presented the quantitative fixed-point reasoning landscape for stochastic systems modeled as probabilistic programs, probabilistic transition systems, and stochastic hybrid systems. This landscape outlines core techniques and their relations for verifying, refuting, or synthesizing upper bounds, lower bounds, or exact values for least fixed points of monotonic operators. Our landscape shall be viewed as pointers – rather than a comprehensive literature review – to navigate interested readers through the vibrantly developing field of fixed-point reasoning in stochastic systems, with many open challenges to explore.

Acknowledgments. We dedicate this article to our dear colleague Martin Fränzle on the occasion of his 60th birthday. Martin has been, amongst many of his other roles, an accomplished scientist devoted to the development of fundamental theories and tools in a variety of fields including (probabilistic) hybrid systems and (stochastic) constraint solving; an academic advisor having inspired many of his young colleagues and students in so many ways; and a genial friend of ours being constantly caring, supportive, and encouraging – as one could tell from his poetic lines of seasonal greetings. This work has been partially funded by the ZJNSF Major Program (No. LD24F020013), by the Fundamental Research Funds for the Central Universities of China (No. 226-2024-00140), and by the ZJU Education Foundation's Qizhen Talent program.

References

1. Abramsky, S., Jung, A.: Domain theory. In: Handbook of Logic in Computer Science, vol. 3: Semantic Structures. Clarendon Press (1994)
2. Alghamdi, M.A., Shahzad, N., Valero, O.: Fixed point theorems in generalized metric spaces with applications to computer science. Fixed Point Theory and Applications **2013**, 1–20 (2013)
3. Baier, C., Hermanns, H.: Weak bisimulation for fully probabilistic processes. In: CAV. Lecture Notes in Computer Science, vol. 1254, pp. 119–130. Springer (1997)
4. Baier, C., Katoen, J.: Principles of model checking. MIT Press (2008)
5. 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)
6. Baldan, P., Eggert, R., König, B., Padoan, T.: Fixpoint theory - upside down. Log. Methods Comput. Sci. **19**(2) (2023)
7. Baldan, P., König, B., Mika-Michalski, C., Padoan, T.: Fixpoint games on continuous lattices. Proc. ACM Program. Lang. **3**(POPL), 26:1–26:29 (2019)
8. Banach, S.: Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales. Fundamenta mathematicae **3**(1), 133–181 (1922)
9. Bao, J., Trivedi, N., Pathak, D., Hsu, J., Roy, S.: Data-driven invariant learning for probabilistic programs. In: CAV (1). LNCS, vol. 13371, pp. 33–54. Springer (2022)
10. Bao, J., Trivedi, N., Pathak, D., Hsu, J., Roy, S.: Data-driven invariant learning for probabilistic programs. Formal Methods in System Design pp. 1–29 (2024)
11. Barthe, G., Katoen, J., Silva, A. (eds.): Foundations of Probabilistic Programming. Cambridge University Press (2020)
12. Batz, K., Chen, M., Junges, S., Kaminski, B.L., Katoen, J., Matheja, C.: Probabilistic program verification via inductive synthesis of inductive invariants. In: TACAS (2). Lecture Notes in Computer Science, vol. 13994, pp. 410–429. Springer (2023)
13. Batz, K., Chen, M., Kaminski, B.L., Katoen, J., Matheja, C., Schröer, P.: Latticed k-induction with an application to probabilistic programs. In: CAV (2). Lecture Notes in Computer Science, vol. 12760, pp. 524–549. Springer (2021)
14. Batz, K., Kaminski, B.L., Matheja, C., Winkler, T.: J-P: MDP. FP. PP - characterizing total expected rewards in markov decision processes as least fixed points with an application to operational semantics of probabilistic programs. In: Principles of Verification (1). Lecture Notes in Computer Science, vol. 15260, pp. 255–302. Springer (2024)
15. Bauer, M.S., Mathur, U., Chadha, R., Sistla, A.P., Viswanathan, M.: Exact quantitative probabilistic model checking through rational search. In: FMCAD. pp. 92–99. IEEE (2017)
16. Bertsekas, D.: Abstract dynamic programming. Athena Scientific (2022)
17. Bhat, S., Agarwal, A., Vuduc, R.W., Gray, A.G.: A type theory for probability density functions. In: POPL. pp. 545–556. ACM (2012)
18. Bhat, S., Borgström, J., Gordon, A.D., Russo, C.V.: Deriving probability density functions from probabilistic functional programs. Log. Methods Comput. Sci. **13**(2) (2017)
19. Biere, A.: Bounded model checking. In: Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185, pp. 457–481. IOS Press (2009)

20. Biere, A., Cimatti, A., Clarke, E.M., Zhu, Y.: Symbolic model checking without bdds. In: TACAS. Lecture Notes in Computer Science, vol. 1579, pp. 193–207. Springer (1999)
21. 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)
22. Bujorianu, L.M.: Stochastic reachability analysis of hybrid systems. Springer Science & Business Media (2012)
23. Calude, C.S., Jain, S., Khoussainov, B., Li, W., Stephan, F.: Deciding parity games in quasipolynomial time. In: STOC. pp. 252–263. ACM (2017)
24. Cassandras, C.G., Lygeros, J. (eds.): Stochastic hybrid systems. CRC Press (2018)
25. Chakarov, A., Sankaranarayanan, S.: Probabilistic program analysis with martingales. In: CAV. Lecture Notes in Computer Science, vol. 8044, pp. 511–526. Springer (2013)
26. Chakarov, A., Voronin, Y., Sankaranarayanan, S.: Deductive proofs of almost sure persistence and recurrence properties. In: TACAS. Lecture Notes in Computer Science, vol. 9636, pp. 260–279. Springer (2016)
27. Chatterjee, K., Fu, H., Goharshady, A.K.: Termination analysis of probabilistic programs through positivstellensatz's. In: CAV (1). Lecture Notes in Computer Science, vol. 9779, pp. 3–22. Springer (2016)
28. Chen, M., Katoen, J., Klinkenberg, L., Winkler, T.: Does a program yield the right distribution? - verifying probabilistic programs via generating functions. In: CAV (1). Lecture Notes in Computer Science, vol. 13371, pp. 79–101. Springer (2022)
29. Chen, Y., Hong, C., Wang, B., Zhang, L.: Counterexample-guided polynomial loop invariant generation by lagrange interpolation. In: CAV (1). Lecture Notes in Computer Science, vol. 9206, pp. 658–674. Springer (2015)
30. Chiang, D., McDonald, C., Shan, C.: Exact recursive probabilistic programming. Proc. ACM Program. Lang. **7**(OOPSLA1), 665–695 (2023)
31. Chutinan, A., Krogh, B.H.: Computational techniques for hybrid system verification. IEEE Trans. Autom. Control. **48**(1), 64–75 (2003)
32. Clarke, E.M., Biere, A., Raimi, R., Zhu, Y.: Bounded model checking using satisfiability solving. Formal Methods Syst. Des. **19**(1), 7–34 (2001)
33. Cousot, P., Cousot, R.: Constructive versions of Tarski's fixed point theorems. Pacific Journal of Mathematics **82**(1), 43–57 (1979)
34. Dahlqvist, F., Silva, A., Kozen, D.: Semantics of probabilistic programming: A gentle introduction. In: Barthe, G., Katoen, J.P., Silva, A. (eds.) Foundations of Probabilistic Programming, pp. 1–42. Cambridge University Press (2020)
35. Davis, M.H.: Markov models & optimization. Routledge (2018)
36. Dijkstra, E.W.: A Discipline of Programming. Prentice-Hall (1976)
37. Feng, S., Chen, M., Su, H., Kaminski, B.L., Katoen, J., Zhan, N.: Lower bounds for possibly divergent probabilistic programs. Proc. ACM Program. Lang. **7**(OOPSLA1), 696–726 (2023)
38. Feng, S., Chen, M., Xue, B., Sankaranarayanan, S., Zhan, N.: Unbounded-time safety verification of stochastic differential dynamics. In: CAV (2). Lecture Notes in Computer Science, vol. 12225, pp. 327–348. Springer (2020)
39. Feng, S., Yang, T., Chen, M., Zhan, N.: A unified framework for quantitative analysis of probabilistic programs. In: Principles of Verification (1). Lecture Notes in Computer Science, vol. 15260, pp. 230–254. Springer (2024)

40. Feng, Y., Zhang, L., Jansen, D.N., Zhan, N., Xia, B.: Finding polynomial loop invariants for probabilistic programs. In: ATVA. Lecture Notes in Computer Science, vol. 10482, pp. 400–416. Springer (2017)
41. Fränzle, M., Chen, M., Kröger, P.: In memory of oded maler: automatic reachability analysis of hybrid-state automata. ACM SIGLOG News **6**(1), 19–39 (2019)
42. Fränzle, M., Hahn, E.M., Hermanns, H., Wolovick, N., Zhang, L.: Measurability and safety verification for stochastic hybrid systems. In: HSCC. pp. 43–52. ACM (2011)
43. Gehr, T., Misailovic, S., Vechev, M.T.: PSI: exact symbolic inference for probabilistic programs. In: CAV (1). Lecture Notes in Computer Science, vol. 9779, pp. 62–83. Springer (2016)
44. Gordon, A.D., Henzinger, T.A., Nori, A.V., Rajamani, S.K.: Probabilistic programming. In: FOSE. pp. 167–181. ACM (2014)
45. Haddad, S., Monmege, B.: Reachability in mdps: Refining convergence of value iteration. In: RP. Lecture Notes in Computer Science, vol. 8762, pp. 125–137. Springer (2014)
46. Haddad, S., Monmege, B.: Interval iteration algorithm for mdps and imdps. Theor. Comput. Sci. **735**, 111–131 (2018)
47. Hark, M., Kaminski, B.L., Giesl, J., Katoen, J.: Aiming low is harder: induction for lower bounds in probabilistic program verification. Proc. ACM Program. Lang. **4**(POPL), 37:1–37:28 (2020)
48. Hartmanns, A., Hermanns, H.: The modest toolset: An integrated environment for quantitative modelling and verification. In: TACAS. Lecture Notes in Computer Science, vol. 8413, pp. 593–598. Springer (2014)
49. 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)
50. Hartmanns, A., Kaminski, B.L.: Optimistic value iteration. In: CAV (2). Lecture Notes in Computer Science, vol. 12225, pp. 488–511. Springer (2020)
51. Hu, J., Lygeros, J., Sastry, S.: Towards a theory of stochastic hybrid systems. In: HSCC. Lecture Notes in Computer Science, vol. 1790, pp. 160–173. Springer (2000)
52. Huang, Z., Dutta, S., Misailovic, S.: Automated quantized inference for probabilistic programs with AQUA. Innov. Syst. Softw. Eng. **18**(3), 369–384 (2022)
53. Jones, C.: Probabilistic NonśDeterminism. Ph.D. thesis, Ph. D. Dissertation. University of Edinburgh, UK (1990)
54. Julius, A.A., Girard, A., Pappas, G.J.: Approximate bisimulation for a class of stochastic hybrid systems. In: ACC. pp. 1–6. IEEE (2006)
55. Jurdzinski, M.: Small progress measures for solving parity games. In: STACS. Lecture Notes in Computer Science, vol. 1770, pp. 290–301. Springer (2000)
56. Kaminski, B.L.: Advanced weakest precondition calculi for probabilistic programs. Ph.D. thesis, RWTH Aachen University (2019)
57. Kaminski, B.L., Katoen, J., Matheja, C.: On the hardness of analyzing probabilistic programs. Acta Informatica **56**(3), 255–285 (2019)
58. Kaminski, B.L., Katoen, J., Matheja, C., Olmedo, F.: Weakest precondition reasoning for expected run-times of probabilistic programs. In: ESOP. Lecture Notes in Computer Science, vol. 9632, pp. 364–389. Springer (2016)
59. Kaminski, B.L., Katoen, J., Matheja, C., Olmedo, F.: Weakest precondition reasoning for expected runtimes of randomized algorithms. J. ACM **65**(5), 30:1–30:68 (2018)

60. Katoen, J., McIver, A., Meinicke, L., Morgan, C.C.: Linear-invariant generation for probabilistic programs: - automated support for proof-based methods. In: SAS. Lecture Notes in Computer Science, vol. 6337, pp. 390–406. Springer (2010)
61. Klinkenberg, L., Batz, K., Kaminski, B.L., Katoen, J., Moerman, J., Winkler, T.: Generating functions for probabilistic programs. In: LOPSTR. Lecture Notes in Computer Science, vol. 12561, pp. 231–248. Springer (2020)
62. Klinkenberg, L., Blumenthal, C., Chen, M., Haase, D., Katoen, J.: Exact bayesian inference for loopy probabilistic programs using generating functions. Proc. ACM Program. Lang. **8**(OOPSLA1), 923–953 (2024)
63. Knaster, B.: Un theoreme sur les functions d'ensembles. Ann. Soc. Polon. Math. **6**, 133–134 (1928)
64. Kozen, D.: Semantics of probabilistic programs. J. Comput. Syst. Sci. **22**(3), 328–350 (1981)
65. Kozen, D.: A probabilistic PDL. In: STOC. pp. 291–297. ACM (1983)
66. Kretínský, J., Meggendorfer, T., Weininger, M.: Stopping criteria for value iteration on stochastic games with quantitative objectives. In: LICS. pp. 1–14. IEEE (2023)
67. 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)
68. Larsen, K.G., Skou, A.: Bisimulation through probabilistic testing. In: POPL. pp. 344–352. ACM Press (1989)
69. Lasserre, J.B.: Moments, Positive Polynomials and Their Applications, vol. 1. World Scientific (2010)
70. Lassez, J., Nguyen, V.L., Sonenberg, L.: Fixed point theorems and semantics: A folk tale. Inf. Process. Lett. **14**(3), 112–116 (1982)
71. Lavaei, A., Soudjani, S., Abate, A., Zamani, M.: Automated verification and synthesis of stochastic hybrid systems: A survey. Autom. **146**, 110617 (2022)
72. Lu, J., Xu, M.: Bisection value iteration. In: APSEC. pp. 109–118. IEEE (2022)
73. Mardare, R., Panangaden, P., Plotkin, G.D.: Quantitative algebraic reasoning. In: LICS. pp. 700–709. ACM (2016)
74. Mardare, R., Panangaden, P., Plotkin, G.D.: Fixed-points for quantitative equational logics. In: LICS. pp. 1–13. IEEE (2021)
75. McIver, A., Morgan, C.: Abstraction, Refinement and Proof for Probabilistic Systems. Monographs in Computer Science, Springer (2005)
76. van de Meent, J., Paige, B., Yang, H., Wood, F.: An introduction to probabilistic programming. CoRR **abs/1809.10756** (2018)
77. Meyn, S.P., Tweedie, R.L.: Markov Chains and Stochastic Stability. Communications and Control Engineering Series, Springer (1993)
78. Moosbrugger, M., Stankovic, M., Bartocci, E., Kovács, L.: This is the moment for probabilistic loops. Proc. ACM Program. Lang. **6**(OOPSLA2), 1497–1525 (2022)
79. Mundici, D.: MV-algebras. A short tutorial (2007)
80. Narayanan, P., Carette, J., Romano, W., Shan, C., Zinkov, R.: Probabilistic inference by program transformation in hakaru (system description). In: FLOPS. Lecture Notes in Computer Science, vol. 9613, pp. 62–79. Springer (2016)
81. Ngo, V.C., Carbonneaux, Q., Hoffmann, J.: Bounded expectations: resource analysis for probabilistic programs. In: PLDI. pp. 496–512. ACM (2018)
82. Øksendal, B., Øksendal, B.: Stochastic differential equations. Springer (2003)
83. Park, D.: Fixpoint induction and proofs of program properties. Machine intelligence **5** (1969)

84. Prajna, S., Jadbabaie, A., Pappas, G.J.: A framework for worst-case and stochastic safety verification using barrier certificates. IEEE Trans. Autom. Control. **52**(8), 1415–1428 (2007)
85. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley Series in Probability and Statistics, Wiley (1994)
86. Quatmann, T., Katoen, J.: Sound value iteration. In: CAV (1). Lecture Notes in Computer Science, vol. 10981, pp. 643–661. Springer (2018)
87. Raskin, J., Chatterjee, K., Doyen, L., Henzinger, T.A.: Algorithms for omega-regular games with imperfect information. Log. Methods Comput. Sci. **3**(3) (2007)
88. Santoyo, C., Dutreix, M., Coogan, S.: Verification and control for finite-time safety of stochastic systems via barrier functions. In: CCTA. pp. 712–717. IEEE (2019)
89. Santoyo, C., Dutreix, M., Coogan, S.: A barrier function approach to finite-time stochastic system verification and control. Autom. **125**, 109439 (2021)
90. Scott, D.: Outline of a mathematical theory of computation. Oxford University Computing Laboratory (1970)
91. Scott, D.S.: Continuous lattices. Toposes, algebraic geometry and logic **274**, 97–136 (1972)
92. Scott, D.S.: Domains for denotational semantics. In: ICALP. Lecture Notes in Computer Science, vol. 140, pp. 577–613. Springer (1982)
93. Scott, D.S.: A type-theoretical alternative to ISWIM, CUCH, OWHY. Theor. Comput. Sci. **121**(1&2), 411–440 (1993)
94. Sheeran, M., Singh, S., Stålmarck, G.: Checking safety properties using induction and a SAT-solver. In: FMCAD. Lecture Notes in Computer Science, vol. 1954, pp. 108–125. Springer (2000)
95. Steinhardt, J., Tedrake, R.: Finite-time regional verification of stochastic non-linear systems. Int. J. Robotics Res. **31**(7), 901–923 (2012)
96. Sutton, R.S.: Reinforcement learning: An introduction. A Bradford Book (2018)
97. Tarski, A.: A lattice-theoretical fixpoint theorem and its applications. (1955)
98. Van Der Schaft, A.J., Schumacher, H.: An introduction to hybrid dynamical systems, vol. 251. springer (2007)
99. Wang, D., Hoffmann, J., Reps, T.W.: Central moment analysis for cost accumulators in probabilistic programs. In: PLDI. pp. 559–573. ACM (2021)
100. Wang, J., Sun, Y., Fu, H., Chatterjee, K., Goharshady, A.K.: Quantitative analysis of assertion violations in probabilistic programs. In: PLDI. pp. 1171–1186. ACM (2021)
101. Watanabe, K., van der Vegt, M., Junges, S., Hasuo, I.: Compositional value iteration with pareto caching. In: CAV (3). Lecture Notes in Computer Science, vol. 14683, pp. 467–491. Springer (2024)
102. Winskel, G.: The formal semantics of programming languages: an introduction. MIT press (1993)
103. Xue, B., Zhan, N., Fränzle, M.: Reach-avoid analysis for polynomial stochastic differential equations. IEEE Trans. Autom. Control. **69**(3), 1882–1889 (2024)
104. Zaiser, F., Murawski, A.S., Ong, C.L.: Guaranteed bounds on posterior distributions of discrete probabilistic programs with loops. Proc. ACM Program. Lang. **9**(POPL), 1104–1135 (2025)