Expectation Invariants for Probabilistic Program Loops as Fixed Points

Aleksandar Chakarov and Sriram Sankaranarayanan

Department of Computer Science
University of Colorado, Boulder, CO
{firstname.lastname}@colorado.edu

Abstract. We present static analyses for probabilistic loops using *expectation invariants*. Probabilistic loops are imperative while-loops augmented with calls to random variable generators. Whereas, traditional program analysis uses Floyd-Hoare style invariants to over-approximate the set of reachable states, our approach synthesizes invariant inequalities involving the expected values of program expressions at the loop head. We first define the notion of expectation invariants, and demonstrate their usefulness in analyzing probabilistic program loops. Next, we present the set of expectation invariants for a loop as a fixed point of the pre-expectation operator over sets of program expressions. Finally, we use existing concepts from abstract interpretation theory to present an iterative analysis that synthesizes expectation invariants for probabilistic program loops. We show how the standard polyhedral abstract domain can be used to synthesize expectation invariants for probabilistic programs, and demonstrate the usefulness of our approach on some examples of probabilistic program loops.

1 Introduction

Inductive loop invariants are commonly used in program verification to prove properties of loops in (non-deterministic) programs. Abstract interpretation provides a powerful framework to synthesize inductive invariants automatically from the given program text [7]. In this paper, we provide a static analysis framework for probabilistic loops that can call random number generators to sample from pre-specified distributions such as *Bernoulli*, *uniform* and *normal*. Probabilistic programs arise in a variety of domains ranging from biological systems [16] to randomized algorithms [21]. In this paper, we present an abstract interpretation framework for deriving *expectation invariants* of probabilistic loops. Expectation invariants are expressions whose expectations *at any given iteration of the loop* exist, and are always non-negative.

Proving expectation invariants often requires approximating the distribution of states after n steps of loop execution (see [2,18,20,9,15] for techniques that approximate distributions in a sound manner). However, even simple programs, such as the program shown in Figure 1, can exhibit complex distributions of reachable states after just a few steps of loop execution (see Figure 2). Extrapolating from a few to arbitrarily many loop iterations requires the notion of "inductive invariants" for probabilistic programs. In this paper, we build upon the standard notion of *quantitative invariants* originally considered by McIver and Morgan [17]. First we extend quantitative invariants from single expressions to a set of expressions that are mutually invariant: multiple expressions

M. Müller-Olm and H. Seidl (Eds.): SAS 2014, LNCS 8723, pp. 85–100, 2014.

whose expectations are nonnegative simultaneously. Next, we characterize invariants as a fixed point, making them amenable to automatic approximation using abstract interpretation. We demonstrate polyhedral analysis over numerical probabilistic programs that manipulate real- and integer-valued state variables.

Our approach first defines the notion of inductive invariants using the pre-expectation operator, along the lines of McIver and Morgan [17]. We lift the pre-expectation operator to a cone of expressions, and subsequently construct a monotone operator over finitely generated cones. Any pre-fixed point of this monotone operator is shown to correspond to expectation invariants. We then use the descending abstract Kleene iteration starting from the cone $\top$ of all affine (or fixed degree polynomial expressions) to iteratively apply the monotone operator to this cone and obtain a pre-fixed point. A (dual) widening operator is used to accelerate this process.

We apply our technique to some small but complex examples of probabilistic programs and demonstrate the power of our approach to synthesize expectation invariants that are otherwise hard to realize manually. We also compare our approach with the tool PRINSYS that synthesizes quantitative invariants using a constraint-based approach by solving constraints on the unknown coefficients of a template invariant form [13,11].

Related Work. The broader area of probabilistic program analysis has seen much progress over the recent past. Our previous work combining symbolic execution of probabilistic programs with volume computation, provides an extensive review of approaches in this area [22]. Therefore, we restrict ourselves to very closely related works.

McIver and Morgan were among the first to consider deductive approaches for probabilistic programs using the concept of *quantitative invariants* [17]. Their work focuses on programs where the stochastic inputs are restricted to discrete distributions over a finite set of support. We naturally lift this restriction to consider a richer class of distributions in this paper including Gaussian, Poisson, Uniform or Exponential random variables. Our setup can use any distributions whose expectations (and some higher moments) exist, and are available. Furthermore, our technique synthesizes invariants that are polynomial expressions involving the program variables. In particular, *indicator functions* over program assertions are not considered in this paper [13,17]. Indicator functions complicate the computation of the pre-expectation when a richer class of distributions are allowed. Finally, McIver & Morgan treat demonic non-deterministic as well as stochastic inputs. Our approach, currently, does not support (demonic) non-determinism; but is potentially extensible when demonic non-determinism is present. Our previous work [3] first considered the relationship between quantitative invariants and the well-known concept of martingales and super-martingales from probability theory [24]. In particular, it demonstrates the use of concentration of measure inequalities to prove probability bounds on assertions at various points in the program [10]. The notion of inductive expectation invariants is a strict generalization of that considered in our previous work. While martingales and super-martingales are analogous to a single inductive linear inequality, we consider the analog of multiple *mutually inductive* linear invariants. The use of abstract interpretation framework is an additional contribution. The generation of quantitative invariants was first studied by Katoen et al. [13], using a constraint-based approach [6,23], implemented in the tool PRINSYS [11]. An experimental comparison is provided in Section 5.

```
real x := rand(-5,3)
real y := rand(-3,5)
int count := 0
while (x+y <= 10)
    if flip(3/4)
      x := x + rand(0,2)
      y := y + 2
    count++
```

```
real x := rand(-5,3)
real y := rand(-3,5)
int count := 0
while (forever)
  if (x + y <= 10)
    if flip(3/4)
       x := x + rand(0,2)
       y := y + 2
    count++
  else
    // Preserve x,y,count
```

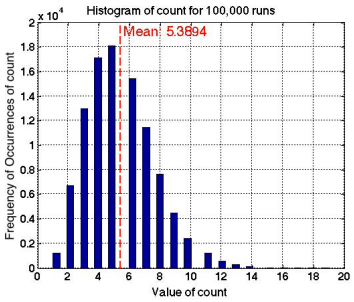

Fig. 1. (**Left**) Simple example of a probabilistic program loop, (**Middle**) Modified loop with *stuttering* semantics, and (**Right**) histogram of the value of count after executing the stuttering loop for at most 25 steps.

Abstract domains for probabilistic programs were first considered by Monniaux [18], by enriching standard abstract domains with bounds on the measure. Refinements of this idea appear in the work of Mardziel et al [15] and Bouissou et al. [2]. Instead of the explicit representations of distributions found in these works, we characterize sets of distributions by means of bounds on moments of expressions. Alternatively, Monniaux presents a backward abstract interpretation scheme to compute the probability of an observable assertion at the program output, and characterize the output distribution [19]. The backwards approach treats the program as a measurable function, and the backward abstract interpretation follows the natural definition of the output distribution through the inverse mapping [5]. However, the approach seemingly requires a user generated query or a systematic gridding of the output states to define the distribution. Cousot and Monerau [9] present a systematic and general abstract semantics for probabilistic programs that views the abstract probabilistic semantics obtained by separately considering abstractions of the program semantics, the probability (event) space, and a "law abstraction" that is a function mapping abstract states to the distribution over the set of possible abstract next states obtained from a single step of program execution. Their approach conveniently captures existing techniques as instances of their framework, while providing new ways of abstracting probabilistic program semantics. Based on our current understanding, the approach in this paper fits into their framework by viewing expectation invariants as representing sets of distributions, and the proposed transfer functions as *law abstractions* that characterize next state distributions.

Example 1. Figure 1 shows a simple probabilistic program written in an imperative language. Each execution of the loop updates variables x,y with probability $\frac{3}{4}$ or chooses to leave them unchanged with probability $\frac{1}{4}$. The variable count acts as a loop counter. Our approach first rewrites the program to yield a *stuttering loop* (see Fig. 1(**Middle**)). Analyzing the stuttering loop yields *expectation invariants* such as

$$(\forall\, n \in \mathbb{N})\ \mathbb{E}(\text{count} \mid n) \leq \frac{56}{9}\,.$$

Here, n refers to the number of iterations of the stuttered loop and $\mathbb{E}(\text{count} \mid n)$ is the expected value of count over the distribution of reachable states after $n \in \mathbb{N}$ iterations.

We ask a natural followup question: what is the expected number of steps the program takes to complete execution, i.e. what is the value $\mathbb{E}(count)$ upon termination of the original program? A simple dynamic approach is to simulate (execute) the program a large number of times and obtain an empirical estimate for $\mathbb{E}(count)$. Figure 1(**Right**) presents the simulation results in the form of a histogram.

Here, we propose a static analysis approach whose goal is to establish facts about the behavior of the program. For one, we can conclude that the original program terminates almost surely since the $\mathbb{E}(\text{count} \mid n)$ is shown to be finite for all n. Knowing that count is always nonnegative, we can now apply Markov's *concentration of measure* inequality [5,10] to conclude bounds on the probabilities of the value of count at any program step: $\mathbb{P}(\text{count} \geq 25 \mid n) \leq \frac{\mathbb{E}(\text{count} \mid n)}{25} \leq \frac{56}{175} \approx 0.32$. Often, we can use much stronger inequalities, should the necessary conditions for these be met. In addition, our analysis yields many other interesting results, for instance: $\forall\, n \in \mathbb{N},\ \mathbb{E}(3\text{count} - 2\text{y} + 2 \mid n) = 0$ and $\mathbb{E}(4\text{x} + 4\text{y} - 9\text{count} \mid n) = 0$.

Outline. The remainder of this paper is organized as follows: Section 2 introduces the preliminaries of probabilistic programs before we extend the discussion to expectation invariants and cones in Section 3. Section 4 presents an abstract interpretation based iterative approach to compute fixed points under the pre-expectation operator. Section 5 is a summary of the experiments we conducted using our prototype version of the tool and a comparison with the PRINSYS tool. Proofs of our main results and details of our probabilistic benchmarks have been provided in an extended version [4].

2 Preliminaries

Probabilistic Programs. Let $\mathcal{P}$ be a probabilistic program in an imperative language with random number generators including `unifInt(lb, ub)`, `unifReal(lb, ub)`, and `gaussian(mean, var)`. These constructs draw values from standard distributions with well-defined, finite expected values. Let $X = \{x_1, \ldots, x_m\}$ be a set of real-valued *program variables* and $R = \{r_1, \ldots, r_l\}$ be a set of real-valued *random variables*. Vectors $\boldsymbol{x}$ and $\boldsymbol{r}$ denote valuations of all program, respectively random, variables. The random variables have a joint distribution $\mathcal{D}_R$. Formally, the distribution is defined over an underlying σ-algebra $(\Omega, \mathcal{F})$ with an appropriate measure μ_r.

A linear inequality over X is an expression of the form $\boldsymbol{a}^T\, \boldsymbol{x} \leq b$ for a vector $\boldsymbol{a} \in \mathbb{R}^m, b \in \mathbb{R}$. A linear assertion $\varphi[X]$ involving X is a conjunction of linear inequalities $\varphi : \bigwedge_{i=1}^{n} \boldsymbol{a}_i^T \boldsymbol{x} \leq b_i$ and can be succinctly expressed in matrix notation as $\varphi : A\boldsymbol{x} \leq \boldsymbol{b}$.

Definition 1 (Probabilistic Loops). *A* probabilistic loop *is a tuple $\langle \mathcal{T}, \mathcal{D}_0, n \rangle$, wherein $\mathcal{T} : \{\tau_1, \ldots, \tau_k\}$ is a set of probabilistic transitions (from the loop head to itself), $\mathcal{D}_0$ is the* initial probability distribution *and n is a formal* loop counter variable.

Each probabilistic transition $\tau_i : \langle \mathbf{g}_i, \mathcal{F}_i \rangle$ consists of (a) guard *assertion $\mathbf{g}_i[X]$ over X; and (b)* update function *$\mathcal{F}_i(\boldsymbol{x}, \boldsymbol{r})$ that yields the next state $\boldsymbol{x}' := \mathcal{F}_i(\boldsymbol{x}, \boldsymbol{r})$.*

In this paper, we restrict ourselves to *piecewise linear* (PWL) probabilistic programs, wherein each transition τ_i has linear assertion guards and piecewise linear updates. Further, we also restrict ourselves to studying expectation invariants over simple loops.

An extension of these ideas to programs with arbitrary control flow structure including nested loops will be discussed in our extended version [4].

Definition 2 (PWL Transitions). *A* piecewise linear *transition* $\tau : \langle \mathbf{g}, \mathcal{F}(\boldsymbol{x}, \boldsymbol{r}) \rangle$ *has the following special structure:*

- $\mathbf{g}$ *is a* linear *guard assertion over* X*;*
- $\mathcal{F}(\boldsymbol{x}, \boldsymbol{r})$ *is a (continuous)* piecewise linear *update function for* X*, where, for ease of presentation,* $\boldsymbol{r}$ *is decomposed into a vector of* continuous (random) choices $\boldsymbol{r}_c$ *and a vector of* discrete Bernoulli choices (coin flips) $\boldsymbol{r}_b$*. As a result, the update function may be written as*

$$F(\boldsymbol{x}, \boldsymbol{r}) = \begin{cases} \mathsf{f}_1 : A_1\boldsymbol{x} + B_1\boldsymbol{r}_c + d_1, & \textit{with probability } p_1, \\ \qquad\vdots & \\ \mathsf{f}_k : A_k\boldsymbol{x} + B_k\boldsymbol{r}_c + d_k, & \textit{with probability } p_k, \end{cases}$$

Options $\mathsf{f}_1, \ldots, \mathsf{f}_k$*, abstract the effect of the Bernoulli choices in* $\boldsymbol{r}_b$*, and are called* forks*, while* $p_1, \ldots, p_k$ *are* fork probabilities *satisfying* $0 < p_i \leq 1$*, and* $\sum_{i=1}^{k} p_i = 1$*.* $A_1, \ldots, A_k \in \mathbb{R}^{m \times m}$*,* $B_1, \ldots, B_k \in \mathbb{R}^{m \times l}$*, and* $d_1, \ldots, d_k \in \mathbb{R}$*.*

No Nondeterminism. For a probabilistic loop $\langle \mathcal{T}, \mathcal{D}_0, n \rangle$, we preclude demonic nondeterminism using two restrictions:

Mutual Exclusion: For all pairs $\tau_1 : \langle \mathbf{g}_1, \mathcal{F}_1 \rangle$ and $\tau_2 : \langle \mathbf{g}_2, \mathcal{F}_2 \rangle$ in $\mathcal{T}$, $\mathbf{g}_1 \wedge \mathbf{g}_2 \equiv false$.
Exhaustiveness: For all transitions τ_i, $\bigvee_{\tau_i \in \mathcal{T}} \mathbf{g}_i \equiv true$.

Mutual exclusion and mutual exhaustiveness together guarantee that precisely one transition can be taken at a time step n and the choice is a function of the state $\boldsymbol{x}$.

Execution Model. A *state* of the probabilistic loop is a tuple $\langle \boldsymbol{x}, n \rangle$ that provides values for the program variables X and the loop counter n. The state $\langle \boldsymbol{x}_0, 0 \rangle$ is called an *initial state* if $\boldsymbol{x}_0$ is a sample drawn from the initial distribution $\mathcal{D}_0$ and $n = 0$.

Definition 3 (Sample Path). *A* sample path *(or an execution) of the loop is an infinite sequence* $(\boldsymbol{x}_0, 0) \xrightarrow{\tau^{(0)}, \boldsymbol{r}_0} (\boldsymbol{x}_1, 1) \xrightarrow{\tau^{(1)}, \boldsymbol{r}_1} (\boldsymbol{x}_2, 2) \to \cdots \xrightarrow{\tau^{(n-1)}, \boldsymbol{r}_{n-1}} (\boldsymbol{x}_n, n) \to \cdots$*, wherein,* ***(a)*** $(\boldsymbol{x}_0, 0)$ *is a sample from* $\mathcal{D}_0$ *and* ***(b)*** *for each* $i \geq 0$*,* $(\boldsymbol{x}_{i+1}, i+1)$ *is obtained by executing the unique transition* $\tau^{(i)} : (\mathbf{g}_i, \mathcal{F}_i)$ *that is enabled on the state* $(\boldsymbol{x}_i, i)$*. This execution involves a sample from the Bernoulli (discrete) random variables to choose a fork of the transition* $\tau^{(i)}$ *and a choice of the continuous random variables* $\boldsymbol{r}_c$ *to obtain* $\boldsymbol{x}_{i+1} = \mathcal{F}_i(\boldsymbol{x}_i, \boldsymbol{r}_i)$*.*

We demonstrate the definitions above on a simple example.

Example 2. In Figure 1 (**Middle**) we present the *stuttering version* of a simple probabilistic program with a loop, where the initial values of the program variables reaching the loop head come from the joint distribution $\mathcal{D}_0 : \langle \mathrm{x}, \mathrm{y}, \mathrm{count} \rangle \sim U[-5, 3] \times U[-3, 5] \times \{0\}$. This modification adds a new program path that preserves the values of program variables once the loop guard $\mathrm{x} + \mathrm{y} \leq 10$ is violated. The program has two transitions $\mathcal{T} : \{\tau_1, \tau_2\}$, where τ_1 represents the loop body:

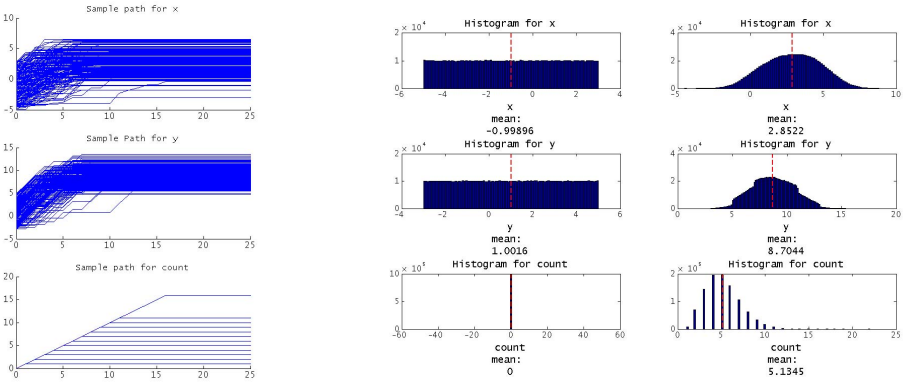

Fig. 2. (**Left**) Some sample paths for the program in Figure 1. (**Right**) Frequency histograms for the distributions $\mathcal{D}_n$ for n = 0, 25.

τ_1 (loop body)	τ_2 (stuttering)
$\mathbf{g}_1 : (x + y \leq 10)$ $\mathcal{F}_{\tau_1} : \begin{cases} f_1 : \begin{bmatrix} \text{x'} & \mapsto \text{x} + r_1, \\ \text{y'} & \mapsto \text{y} + 2, \\ \text{count'} & \mapsto \text{count} + 1, \end{bmatrix} & \text{w.p. } \frac{3}{4} \\ f_2 : \begin{bmatrix} \text{x'} & \mapsto \text{x}, \\ \text{y'} & \mapsto \text{y}, \\ \text{count'} & \mapsto \text{count} + 1, \end{bmatrix} & \text{w.p. } \frac{1}{4} \end{cases}$	$\mathbf{g}_2 : (x + y > 10)$ $\mathcal{F}_{\tau_2} : \begin{cases} \text{x'} & \mapsto \text{x}, \\ \text{y'} & \mapsto \text{y}, \\ \text{count'} & \mapsto \text{count}, \end{cases}$

Here r_1 represents the uniform random variable over $[0, 2]$. Transition τ_2 represents the *stuttering* after $x + y > 10$. It is added to satisfy the mutual exclusiveness and exhaustiveness requirements. It has a single fork that preserves the values of x, y, count. Figure 2 depicts 200 sample paths obtained by simulating the program (for 25 steps) and distributions $\mathcal{D}_n$ for $n = 0$ and $n = 25$ obtained by running the program 10^6 times.

Operator Semantics: Probabilistic program semantics can be thought of as continuous linear operators over over the state distributions, starting from the initial distribution $\mathcal{D}_0$: $\mathcal{D}_0 \xrightarrow{[\![\mathcal{P}]\!]} \mathcal{D}_1 \xrightarrow{[\![\mathcal{P}]\!]} \cdots \xrightarrow{[\![\mathcal{P}]\!]} \mathcal{D}_n \xrightarrow{[\![\mathcal{P}]\!]} \cdots$. Here $[\![\mathcal{P}]\!]$ models the effect of a single loop iteration and $\mathcal{D}_n$ is the distribution of the states after n iterations of the loop. This matches the standard probabilistic program semantics [14,19]. The definition of $\mathcal{D}_n$ and $[\![\mathcal{P}]\!]$ are described in the extended version [4].

Pre-Expectations. We now define the useful concept of pre-expectation of an expression $\mathbf{e}$ over the program variables across a transition τ following earlier work by McIver and Morgan [17]. Let $\tau : \langle \mathbf{g}, \mathcal{F} \rangle$ be a transition and $\mathbf{e}[\boldsymbol{x}]$ be an expression involving the state variables $\boldsymbol{x}$ of the program.

The pre-expectation operator $\texttt{pre}\mathbb{E}_\tau$ is an *expression transformer* that associates each expression e with the next-step *expectation* expression $\texttt{pre}\mathbb{E}_\tau(\mathsf{e}[\boldsymbol{x}'])$ across τ, in terms of the *current state variables of the program*. Formally,

$$\texttt{pre}\mathbb{E}_\tau(\mathsf{e}[\boldsymbol{x}']) : \mathbb{E}_R(\mathsf{e}[\boldsymbol{x}' \mapsto \mathcal{F}(\boldsymbol{x}, \boldsymbol{r})] \mid \boldsymbol{x})$$

The expectation $\mathbb{E}_R$ is taken over the distribution of $\boldsymbol{r}$ in the transition τ.

Consider a PWL transition τ with $k > 0$ forks, $\mathsf{f}_1, \ldots, \mathsf{f}_k$, each of the form $\mathsf{f_j} : A_j\boldsymbol{x} + B_j\boldsymbol{r} + d_j$ with fork probability p_j. The pre-expectation operator is defined as

$$\texttt{pre}\mathbb{E}_\tau(\mathsf{e}') = \sum_{j=1}^{k} p_j \mathbb{E}_R(\text{PRE}(\mathsf{e}', f_j) \mid \boldsymbol{x})$$

where $\text{PRE}(\mathsf{e}', f_j)$ is the substitution of post variables $\boldsymbol{x}'$ for their update values $f_j(\boldsymbol{x}, \boldsymbol{r})$ in expression e. The expectation $\mathbb{E}_R(g)$ denotes the expectation of g over the joint distribution $\mathcal{R}$ of the random variables.

Example 3. We illustrate the notion of a pre-expectation of a program expression by considering the expression $3 + 2\mathrm{x} - \mathrm{y}$ across transition τ_1 in the Figure 1.

$$\texttt{pre}\mathbb{E}_{\tau_1}(3 + 2\mathrm{x}' - \mathrm{y}') : \begin{pmatrix} \frac{3}{4}[3 + 2\mathbb{E}_{r_1}(x + r_1) - (y + 2)] + & // \text{ from fork } f_1 \\ \frac{1}{4}[3 + 2x - y] & // \text{ from fork } f_2 \end{pmatrix}.$$

Simplifying, we obtain $\texttt{pre}\mathbb{E}_{\tau_1}(3 + 2\mathrm{x}' - \mathrm{y}') = 3 + 2x - y + \frac{3}{2}\mathbb{E}_{r_1}(r_1) - \frac{3}{2}$. Noting that $\mathbb{E}_{r_1}(r_1) = 1$, we obtain $\texttt{pre}\mathbb{E}_{\tau_1}(3 + 2\mathrm{x}' - \mathrm{y}') = 3 + 2\mathrm{x} - \mathrm{y}$.

Likewise, we define $\texttt{pre}\mathbb{E}(\mathsf{e}')$ (without a transition as a subscript) as

$$\mathbb{1}_{g_{\tau_1}} \times \texttt{pre}\mathbb{E}_{\tau_1}(\mathbf{e}') + \cdots + \mathbb{1}_{g_{\tau_k}} \times \texttt{pre}\mathbb{E}_{\tau_k}(\mathbf{e}'),$$

wherein $\mathbb{1}_g(\boldsymbol{x})$ is the *indicator* function: $\mathbb{1}_g(\boldsymbol{x}) = 1$ if $\boldsymbol{x} \models g(\boldsymbol{x}),$ and 0, otherwise.

We now state a key result involving pre-expectations. Consider a prefix σ of a sample execution $(\boldsymbol{x}_0, 0) \rightarrow (\boldsymbol{x}_1, 1) \rightarrow \cdots \rightarrow (\boldsymbol{x}_n, n)$. Given that the current state is $(\boldsymbol{x}_n, n)$, we wish to find out the expectation of an expression e over the distribution of *all possible* next states $(\boldsymbol{x}_{n+1}, n + 1)$. Let $\hat{\mathsf{e}} : \texttt{pre}\mathbb{E}(\mathsf{e}')$.

Lemma 1. *The expected value of* e *over the post-state distribution starting from state* $(\boldsymbol{x}_n, n)$ *is the value of the pre-expectation* $\hat{\mathsf{e}}$ *evaluated over the current state* $\boldsymbol{x}_n$*:*

$$\mathbb{E}(\mathsf{e}(\boldsymbol{x}_{n+1})|\boldsymbol{x}_n, n) = \hat{\mathsf{e}}(\boldsymbol{x}_n) = \sum_{\tau_i \in \mathcal{T}} \mathbb{1}_{g_i}(\boldsymbol{x}_n)\texttt{pre}\mathbb{E}_{\tau_i}(\mathsf{e}').$$

Finally, we extend Lemma 1 to the full *distribution* $\mathcal{D}_n$ from which $\boldsymbol{x}_n$ is drawn.

Lemma 2. *The expected value of* e *over the post-state distribution* $\mathcal{D}_{n+1}$ *given a distribution* $\mathcal{D}_n$ *for the current state valuations* $\boldsymbol{x}_n$ *satisfies:*

$$\mathbb{E}_{\mathcal{D}_{n+1}}(\mathsf{e}(\boldsymbol{x}_{n+1})) = \mathbb{E}_{\mathcal{D}_n}(\texttt{pre}\mathbb{E}(\mathsf{e})(\boldsymbol{x}_n)) = \mathbb{E}_{\mathcal{D}_n}[\hat{\mathsf{e}}] = \mathbb{E}_{\mathcal{D}_n}\left[\sum_{\tau_i \in \mathcal{T}} \mathbb{1}_{g_i}(\boldsymbol{x}_n)\texttt{pre}\mathbb{E}_{\tau_i}(\mathsf{e}')\right].$$

3 Expectation Invariants

Expectation invariants are invariant inequalities on the expected value of program expressions. Therefore, one could view the set of possible state distributions $\mathcal{D}_i$ at step i as the concrete domain over which our analysis operates to produce the abstract facts in the form of *expectation invariants* over these distributions. We formalize expectation invariants and derive a fixed point characterization of expectation invariants.

3.1 Definitions and Examples

Let $\mathcal{P} : \langle \mathcal{T}, \mathcal{D}_0, n \rangle$ be a probabilistic loop and let $\langle \boldsymbol{x}_0, 0 \rangle$ be the initial state of the system. From Section 2 we know that $\boldsymbol{x}_0$ is drawn from an initial distribution $\mathcal{D}_0$ and that any n-step sample execution of $\mathcal{P}$ defines a sample trajectory through the distributions of reachable states $\mathcal{D}_0, \ldots, \mathcal{D}_n$ at step i for any $0 \leq i \leq n$. We then define the *expectation* of a program expression e *at time step* n as $\mathbb{E}(\mathsf{e}(\boldsymbol{x}_n) \mid n) = \mathbb{E}_{\mathcal{D}_n}(\mathsf{e}(\boldsymbol{x}_n))$.

Definition 4 (Expectation Invariants). *An* e *over the program variables* X *is called an* expectation invariant *(EI) iff for all* $n \geq 0$, $\mathbb{E}(\mathsf{e} \mid n) \geq 0$.

Thus, expectation invariants are program expressions whose expectations over the initial distribution are non-negative, and under *any* number of iterations of the probabilistic loop remain non-negative.

Example 4. Consider the program from Example 1, and the expression $\mathrm{y} - \mathrm{x}$. Initially, $\mathbb{E}(\mathrm{y} - \mathrm{x} \mid 0) = \mathbb{E}_{\mathcal{D}_0}(\mathrm{y} - \mathrm{x}) = 1 - (-1) = 2 \geq 0$. We can show that $\mathbb{E}(\mathrm{y} - \mathrm{x} \mid i) = \mathbb{E}(\mathrm{y} \mid i) - \mathbb{E}(\mathrm{x} \mid i) \geq 0$ at any step i. Therefore, $\mathrm{y} - \mathrm{x}$ is an expectation invariant.

The concept of expectation invariant defined here is closely related to that of martingales studied in our earlier work [3]. The importance of expectation invariants is that a set of "inductive" EI is a set of mutually inductive constraints over state distributions. This allows the analysis to track and transfer sufficient amount of information about the distributions across loop iterations without ever having to explicitly construct these.

Proving EI. We now focus on the question of proving that a given expression e over the program variables is an expectation invariant. This requires constructing (approximations) to the distribution $\mathcal{D}_n$ for each n, or alternatively, an argument based on mathematical induction. We first observe an important property of each $\mathcal{D}_n$.

Definition 5 (Admissible Distribution). *We say that a distribution* $\mathcal{D}$ *over the state-space* $\mathcal{X}$ *is* admissible *if all moments exist.*[1] *In other words, for any polynomial* $p(\boldsymbol{x})$ *over the program variables,* $\mathbb{E}_{\mathcal{D}}(p(\boldsymbol{x}))$ *exists, and is finite.*

Let us assume that any program $\mathcal{P}$ which we attempt to analyze is such that

[1] While the existence of only the first moment suffices, our experiments demonstrate that our current synthesis approach can be extended to polynomial expectation invariants.

1. $\mathcal{D}_0$, the initial state distribution, is admissible;
2. For each transition τ, the distribution of the random variables $\mathcal{D}_R$ is admissible.

Under these assumptions and following the linearity of the statements and the guards in the program, we can show the following fact.

Lemma 3. *For all $n \in \mathbb{N}$, the distribution $\mathcal{D}_n$ is admissible.*

Rather than construct $\mathcal{D}_n$ explicitly for each n (which quickly becomes impractical), we formulate the principle of inductive expectation invariants. Consider expressions $E = \{\mathtt{e}_1, \ldots, \mathtt{e}_m\}$ wherein each $\mathtt{e}_i$ is a linear (or polynomial) expression involving the program variables.

Definition 6 (Inductive Expectation Invariants). *The set E of expressions forms an inductive expectation invariant of the program $\mathcal{P}$ iff for each $\mathtt{e}_j$, $j \in [1, m]$,*

1. $\mathbb{E}_{\mathcal{D}_0}(\mathtt{e}_j) \geq 0$*, i.e., the expectation at the initial step is non-negative.*
2. *For every admissible distribution $\mathcal{D}$ over the state-space $\mathcal{X}$,*

$$(\mathbb{E}_{\mathcal{D}}(\mathtt{e}_1) \geq 0 \ \wedge \ \cdots \ \wedge \ \mathbb{E}_{\mathcal{D}}(\mathtt{e}_m) \geq 0) \ \models \ \mathbb{E}_{\mathcal{D}}(\mathtt{pre}\mathbb{E}(\mathtt{e}_j)) \geq 0\,. \tag{1}$$

The inductive expectation invariant principle stated above follows the standard Floyd-Hoare approach of "abstracting away" the distribution at the n^{th} step by the inductive invariant itself, and using these to show that the invariant continues to hold for one more step. Furthermore, it abstracts away from a specific $\mathcal{D}_n$ to any admissible distribution $\mathcal{D}$. However, Def. 6 is quite unwieldy, in practice, since the quantification over *all possible admissible distributions* $\mathcal{D}$ over the state space $\mathcal{X}$ is a higher order quantifier (over probability spaces and measurable functions). Rather than reason with this quantifier, we will use the following facts about expectations to formulate a new principle:

Theorem 1 (Facts About Expectations over Admissible Distributions). *The following hold over all possible admissible distributions $\mathcal{D}$ over a σ-algebra $\mathcal{X}$, linear assertion φ, and linear (or polynomial expressions) $\mathtt{e}, \mathtt{e}_1, \ldots, \mathtt{e}_k$:*

1. *Linearity of expectation: $\mathbb{E}_{\mathcal{D}}(\lambda_1\mathtt{e}_1 + \ldots + \lambda_k\mathtt{e}_k) = \lambda_1\mathbb{E}_{\mathcal{D}}(\mathtt{e}_1) + \cdots + \lambda_k\mathbb{E}_{\mathcal{D}}(\mathtt{e}_k)$, for $\lambda_i \in \mathbb{R}$.*
2. *Indicator Functions: $\mathbb{E}_{\mathcal{D}}(\mathbb{1}_{\mathtt{e}\geq 0} \times \mathtt{e}) \geq 0$, and in general, if $\varphi \models \mathtt{e} \geq 0$ then $\mathbb{E}_{\mathcal{D}}(\mathbb{1}_{\varphi} \times \mathtt{e}) \geq 0$, provided $[\![\varphi]\!]$ is measurable.*
3. *$\mathbb{E}_{\mathcal{D}}(\mathbb{1}_{\varphi}\mathtt{e} + \mathbb{1}_{\neg\varphi}\mathtt{e}) = \mathbb{E}_{\mathcal{D}}(\mathtt{e})$, provided $[\![\varphi]\!]$ is measurable.*

Using these facts as "axioms", we attempt to reformulate the key step 2 of Def. 6 as a simple quantified statement in (first-order) linear arithmetic. Consider, once again, the key statement of the principle (1). The central idea of our approach is to express the pre-expectation $\mathtt{pre}\mathbb{E}(\mathtt{e}_j)$ for each $\mathtt{e}_j \in E$ as

$$\mathtt{pre}\mathbb{E}(\mathtt{e}_j) = \sum_{i=1}^{m} \lambda_{j,i}\mathtt{e}_i + \sum_{p} \mu_{j,p}\,(\mathbb{1}_{\varphi_p} \times g_p)\,, \tag{2}$$

wherein $\lambda_{j,i} \geq 0$ and $\mu_{j,p} \geq 0$ are real-valued multipliers, g_p are linear expressions over the program variables and φ_p are assertions such that $\varphi_p \models g_p \geq 0$. The origin of the expressions g_p and assertions φ_p will be made clear, shortly. Let us fix a finite set of expressions $E = \{\mathtt{e}_1, \ldots, \mathtt{e}_m\}$.

Lemma 4. *If E satisfies the relaxed induction principle* (2) *then E satisfies the original induction principle* (1).

3.2 Conic Inductive Expectation Invariants

We now formalize this intuition using the concept of *conic inductive expectation invariants*. Let $\mathcal{P}$ be a program with transitions $\mathcal{T}$. Let $\mathbf{g}_i$ be a linear assertion representing the guard of the transition τ_i. We express $\mathbf{g}_i$ as $\bigwedge_{j=1}^{n_i} g_{i,j} \geq 0$, wherein $g_{i,j}$ are affine program expressions. Let $\boldsymbol{g_i} : (g_{i,1} \ \ldots \ g_{i,n_i})^T$ be a vector representing $\mathbf{g}_i$. Likewise, let $E = \{\mathtt{e}_1, \ldots, \mathtt{e}_m\}$ be a finite set of expressions, we denote the vector of expressions as $\boldsymbol{e} : (\mathtt{e}_1, \ldots, \mathtt{e}_m)^T$.

Definition 7 (Conic Inductive Expectation Invariants). *The finite set E is a* conic inductive invariant *of the program $\mathcal{P}$ iff for each $\mathtt{e}_j \in E$,*

1. *Initial Condition: $\mathbb{E}_{\mathcal{D}_0}(\mathtt{e}_j) \geq 0$*
2. *Induction Step: There exists a vector of multipliers $\boldsymbol{\lambda}_j \geq 0$, such that for each transition $\tau_l : (\mathbf{g}_l, \mathcal{F}_l)$, $\mathtt{pre}\mathbb{E}_{\tau_l}(\mathtt{e}_j)$ can be expressed as a conic combination of expressions in E and the expressions in $\mathbf{g}_l$:*

$$(\exists\, \boldsymbol{\lambda}_j \geq 0)\ (\forall\, \tau_l \in \mathcal{T})\ (\exists\, \boldsymbol{\mu_l} \geq 0)\ \mathtt{pre}\mathbb{E}_{\tau_l}(\mathtt{e}_j) = \boldsymbol{\lambda}_j^T\, \boldsymbol{e} + \boldsymbol{\mu_l}^T \boldsymbol{g_l}\,. \tag{3}$$

In particular, we note that the order of quantification in Eq. (3) is quite important. We note for a given expression $\mathtt{e}_j$ the multipliers $\boldsymbol{\lambda}_j$ must stay the same across all the transitions $\tau_l \in \mathcal{T}$. This will ensure the applicability of the linearity of expectation. Changing the order of quantification makes the rule unsound, as discussed in the extended version of the paper[4].

Example 5. The set $E = \{e_1 : \mathrm{y} - 2\mathrm{x},\ e_2 : 2\mathrm{x} - \mathrm{y} + 3,\ e_3 : 4\mathrm{x} - 3\mathrm{count} + 4,\ e_4 : -2\mathrm{x} + \mathrm{y} - 3,\ e_5 : -4\mathrm{x} + 3\mathrm{count} - 4\}$ is a conic inductive invariant for the program in Example 1. Consider $\mathtt{e}_1 : \mathrm{y} - 2\mathrm{x}$. We have

$$\mathtt{pre}\mathbb{E}_{\tau_1}(\mathtt{e}_1) : \mathbb{E}_{r_1}\left(\frac{3}{4}(\mathrm{y} + 2 - 2\mathrm{x} - 2r_1) + \frac{1}{4}(\mathrm{y} - 2\mathrm{x})\right) = \mathrm{y} - 2\mathrm{x}.$$

Likewise, $\mathtt{pre}\mathbb{E}_{\tau_2}(\mathtt{e}_1) : \mathtt{e}_1$, since τ_2 is a stuttering transition.

Therefore, setting $\boldsymbol{\lambda} : (1\ 0\ 0\ 0\ 0)^T$, we obtain $\mathtt{pre}\mathbb{E}(\mathtt{e}_1) : \boldsymbol{\lambda}^T\boldsymbol{e} + \mathbf{0} \times \mathbb{1}_{x+y\leq 10}$. For a non-trivial example, see the extended version of the paper [4].

Theorem 2. *Let E be a conic inductive invariant for a program $\mathcal{P}$ as given by Definition 7. It follows that each $\mathtt{e}_j \in E$ is an expectation invariant of the program.*

Thus far, we have presented inductive expectation invariants as a finite set of expressions $E = \{\mathtt{e}_1, \ldots, \mathtt{e}_m\}$, satisfying the conditions in Definitions 6 or 7. We transfer our notion from a finite set of expressions to a finitely generated cone of these in preparation for our fixed point characterization given in the next section.

Definition 8 (Cones). *Let $E = \{\mathsf{e}_1, \ldots, \mathsf{e}_k\}$ be a finite set of program expressions over the program variables $\boldsymbol{x}$. The set of* conic combinations (the cone) *of E is defined as*

$$\mathsf{Cone}(E) = \left\{ \lambda_0 + \sum_{i=1}^{k} \lambda_i \mathsf{e}_i \mid 0 \leq \lambda_i, 0 \leq i \leq k \right\}.$$

Expressions e_i are called the generators *of the cone.*

Given a non-empty linear assertion assertion $\varphi : \bigwedge_{i=1}^{k} \mathsf{e}_i \geq 0$, it is well-known that $\varphi \models \mathsf{e} \geq 0$ iff $\mathsf{e} \in \mathsf{Cone}(\mathsf{e}_1, \ldots, \mathsf{e}_k)$. Likewise, let E be an inductive expectation invariant. It follows that any $\mathsf{e} \in \mathsf{Cone}(E)$ is an expectation invariant of the program $\mathcal{P}$.

Example 6. Revisiting Example 5, we consider the conic combination:

$$4(-2\mathrm{x} + \mathrm{y} - 3) + 3(4\mathrm{x} - 3\mathrm{count} + 4) = 4\mathrm{x} + 4\mathrm{y} - 9\mathrm{count}$$

As a result, we conclude that $\mathbb{E}_{\mathcal{D}_n}(4\mathrm{x} + 4\mathrm{y} - 9\mathrm{count}) \geq 0$ at each step $n \geq 0$.

Analyzing the program by replacing the probabilistic statements with non-deterministic choice, and performing polyhedral abstract interpretation yields the invariant $\mathrm{x} + \mathrm{y} \leq 14$ [8]. This allows us to bound the set of support for $\mathcal{D}_n$, and also allows us to conclude that $\mathbb{E}_{\mathcal{D}_n}(14 - \mathrm{x} - \mathrm{y}) \geq 0$. Combining these facts, we obtain,

$$\mathbb{E}_{\mathcal{D}_n}(56 - 9\mathrm{count}) \geq 0, \text{ or equivalently, } \mathbb{E}_{\mathcal{D}_n}(\mathrm{count}) \leq \frac{56}{9}.$$

4 Expectation Invariants as Fixed Points

In this section, we show that the notion of conic invariants as presented in Definition 7 can be expressed as a (pre-) fixed point of a monotone operator over finitely generated cones representing sets of expressions. This naturally allows us to use abstract interpretation starting from the cone representing all expressions ($\top$) and performing a downward Kleene iteration until convergence. We use a (dualized) widening operator to ensure fast convergence to fixed point in finitely many iterations.

Let $\mathcal{P}$ be a program over variables $\boldsymbol{x}$ with transitions $\mathcal{T} : \{\tau_1, \ldots, \tau_k\}$ and initial distribution $\mathcal{D}_0$. For simplicity, we describe our approach to generate affine expressions of the form $c_0 + \boldsymbol{c}^T\boldsymbol{x}$ for $c_0 \in \mathbb{R}, \boldsymbol{c} \in \mathbb{R}^n$. Let $\mathbb{A}(\boldsymbol{x})$ represent the set of all affine expressions over $\boldsymbol{x}$.

Polyhedral Cones of Expectation Invariant Candidates: Our approach uses finitely generated cones $I : \mathsf{Cone}(E)$ where $E = \{\mathsf{e}_1, \ldots, \mathsf{e}_m\}$ is a finite set of affine expressions over $\boldsymbol{x}$. Each element $\mathsf{e} \in I$ represents a *candidate expectation invariant*. Once a (pre-) fixed point is found by our technique, we obtain a cone $I^* : \mathsf{Cone}(E^*)$, wherein E^* will be shown to be a conic inductive invariant according to Definition 7.

A finitely generated cone of affine expressions $I : \mathsf{Cone}(E)$ is represented by a polyhedral cone of its coefficients $C(I) : \{(c_0, \boldsymbol{c}) \mid c_0 + \boldsymbol{c}^T\boldsymbol{x} \in I\}$. The generators of $C(I)$ are coefficient vectors $(c_{0,i}, \boldsymbol{c}_i)$ representing the expression $\mathsf{e}_i : c_{0,i} + \boldsymbol{c}_i^T\boldsymbol{x}$.

Our analysis operates on the lattice of polyhedral cone representations, CONES, ordered by the set theoretic inclusion operator $\subseteq$. This is, in fact, dual to the polyhedral domain, originally proposed by Cousot & Halbwachs [8].

Initial Cone: For simplicity, we will assume that $\mathcal{D}_0$ is specified to us, and we are able to compute $E_{\mathcal{D}_0}(\boldsymbol{x})$ precisely for each program variable. The initial cone I_0 is given by

$$I_0 : \mathsf{Cone}\left(\{x_1 - \mathbb{E}_{\mathcal{D}_0}(x_1), \mathbb{E}_{\mathcal{D}_0}(x_1) - x_1, \cdots, \mathbb{E}_{\mathcal{D}_0}(x_n) - x_n, x_n - \mathbb{E}_{\mathcal{D}_0}(x_n)\}\right) .$$

Such a cone represents the invariant candidates $x_i = \mathbb{E}_{\mathcal{D}_0}(x_i)$.

Pre-Expectation Operators: We now describe the parts of the monotone operator over finitely generated cones. Let $E = \{\mathsf{e}_1, \ldots, \mathsf{e}_m\}$ be a set of expressions. Let $\tau : \langle \mathbf{g}, \mathcal{F} \rangle$ be a transition, wherein $\mathbf{g} : \bigwedge_{l=1}^{p} g_l \geq 0$. We first present a pre-expectation operator over cones, lifting the notation $\mathtt{pre}\mathbb{E}_\tau$ from expressions to cones of such:

Definition 9 (Pre-Expectation Operator). *The* pre-expectation *of a cone* $I : \mathsf{Cone}(E)$ w.r.t a transition τ *is defined as:*

$$\mathtt{pre}\mathbb{E}_\tau(I) = \{(\mathsf{e}, \boldsymbol{\lambda}) \in \mathbb{A}(x) \times \mathbb{R}^m \mid \boldsymbol{\lambda} \geq 0 \wedge \exists\, \boldsymbol{\mu} \geq 0\, (\mathtt{pre}\mathbb{E}_\tau(\mathsf{e}) \equiv \sum_{j=1}^{m} \boldsymbol{\lambda}_j \mathsf{e}_j + \sum_{i=1}^{p} \boldsymbol{\mu}_i g_i)\}.$$

The refinement $\mathtt{pre}\mathbb{E}_\tau(I)$ of a cone contains all affine program expressions whose pre-expectation belongs to the conic hull of I and the cone generated by the guard assertion. For technical reasons, we attach to each expression a certificate $\boldsymbol{\lambda}$ that shows its membership back in the cone. This can be seen as a way to ensure the proper order of quantification in Definition 7.

Given a polyhedron $C(I)$ representing I, we can show that $C(\mathtt{pre}\mathbb{E}_\tau(I))$ is a polyhedral cone over the variables $(c_0, \boldsymbol{c})$ representing the expression coefficients and $\boldsymbol{\lambda}$ for the multipliers. Our extended version [4] presents in detail the steps for computing the pre-expectation of a cone as well as the fixpoint computation across multiple transitions.

Next, we define a pre-expectation operator across all transitions:

$$\mathtt{pre}\mathbb{E}(I) = \{\mathsf{e} \in \mathbb{A}(x) \mid (\exists\, \boldsymbol{\lambda} \geq 0)\, (\mathsf{e}, \boldsymbol{\lambda}) \in \bigcap_{j=1}^{k} \mathtt{pre}\mathbb{E}_{\tau_j}(I)\}$$

An expression e belongs to $\mathtt{pre}\mathbb{E}(I)$ if for some $\boldsymbol{\lambda} \geq 0$, $(e, \boldsymbol{\lambda}) \in \mathtt{pre}\mathbb{E}_{\tau_j}(I)$ for each transition $\tau_j \in \mathcal{T}$.

Given a cone $C(I)$, we first compute the cones $C(\hat{I}_1), \ldots, C(\hat{I}_k)$ representing the pre-expectations across transitions $\tau_1, \ldots, \tau_k$, respectively. Next, we compute $C(I') : (\exists\, \boldsymbol{\lambda}) \bigcap_{j=1}^{k} C(\hat{I}_j)$, representing $I' : \mathtt{pre}\mathbb{E}(I)$, by intersecting the cones $C(\hat{I}_j)$ and projecting the dimensions corresponding to $\boldsymbol{\lambda}$.

We define the operator $\mathcal{G}$ over cones as $\mathcal{G}(I) : I_0 \cap \mathtt{pre}\mathbb{E}(I)$, where I_0 is the initial cone.

Theorem 3. *The operator* $\mathcal{G}$ *satisfies the following properties:*

1. $\mathcal{G}$ *is a monotone operator over the lattice* CONES *ordered by set-theoretic inclusion.*
2. *A finite set of affine expressions* E *is a conic inductive invariant (Def. 7) if and only if* $I : \mathsf{Cone}(E)$ *is a pre-fixed point of* $\mathcal{G}$*, i.e,* $I \subseteq \mathcal{G}(I)$.

Iteration over Polyhedral Cones: Our goal is to compute the greatest fixed point of $\mathcal{G}$ representing the largest cone of expressions whose generators satisfy Definition 7. We implement this by a downward Kleene iteration until we obtain a pre-fixed point, which in the ideal case is also the greatest fixed point of $\mathcal{G}$.

$$(J_0 : \mathbb{A}(x)) \supseteq (J_1 : \mathcal{G}(J_0)) \supseteq \cdots (J_{k+1} : \mathcal{G}(J_k)) \cdots \text{ until convergence: } J_i \subseteq J_{i+1}\,.$$

However, the domain CONES has infinite descending chains and is not a complete lattice. Therefore, the greatest fixed point cannot necessarily be found in finitely many steps by the Kleene iteration. We resort to a *dual widening* operator $\widetilde{\nabla}$ to force convergence of the downward iteration.

Definition 10 (Dual Widening). *Let I_1, I_2 be two successive cone iterates, satisfying $I_1 \supseteq I_2$. The operator $\widetilde{\nabla}(I_1, I_2)$ is a* dual widening operator *if:*

- $\widetilde{\nabla}(I_1, I_2) \subseteq I_1$, $\widetilde{\nabla}(I_1, I_2) \subseteq I_2$*;*
- *For every infinite descending sequence $J_0 \supseteq \mathcal{G}(J_0) \supseteq \mathcal{G}^2(J_0) \supseteq \cdots$, the* widened sequence *$J'_0 = J_0$, $J'_n = J'_{n-1} \widetilde{\nabla} J_n$ converges in finitely many steps.*

A common strategy to compute an approximation of the greatest fixed point when using dual widening is to delay widening for a fixed number K of iterations.

Example 7. Consider a simulation of a peg performing an unbounded random walk in two dimensions (x, y). Starting at the origin, at every step the peg chooses uniformly at random a direction {N, E, S, W} and a random step size $r_1 \sim U[0, 2]$. The program 2D-WALK tracks the steps (count) and the Manhattan distance (dist) to the origin.

The following table summarizes the result of the expectation invariant analysis:

Cone	Generators	Constraints	Cone	Generators	Constraints
I_0	1, $-$**count**, count, x, $-$x, y, $-$y, dist, $-$dist,	$c_0 \geq 0$	I_4	1, $\mathbf{4}-$ **count**, count, x, $-$x, y, $-$y, dist, $-$dist	$c_0 + 4c_4 \geq 0$, $c_0 \geq 0$
I_1	1, $\mathbf{1}-$ **count**, count, x, $-$x, y, $-$y, dist, $-$dist	$c_0 + c_4 \geq 0$, $c_0 \geq 0$	I_5	1, $\mathbf{5}-$ **count**, count, x, $-$x, y, $-$y, dist, $-$dist	$c_0 + 4c_4 \geq 0$, $c_0 \geq 0$
I_2	1, $\mathbf{2}-$ **count**, count, x, $-$x, y, $-$y, dist, $-$dist	$c_0 + 2c_4 \geq 0$, $c_0 \geq 0$	$\vdots$	$\vdots$	$\vdots$
I_3	1, $\mathbf{3}-$ **count**, count, x, $-$x, y, $-$y, dist, $-$dist	$c_0 + 3c_4 \geq 0$, $c_0 \geq 0$	I_∞	1, count, x, $-$x, y, $-$y, dist, $-$dist	$c_4 \geq 0$, $c_0 \geq 0$

The table shows the value of expression count is unbounded from above. To force convergence, we employ dual widening after a predefined number ($K = 5$) of iterations.

Definition 11 (Standard Dual Widening). *Let $I_1 = Cone(g_1, \ldots, g_k)$ and $I_2 = Cone(h_1, \ldots, h_l)$ be two finitely generated cones such that $I_1 \supseteq I_2$. The dual widening operator $I_1 \widetilde{\nabla} I_2$ is defined as $I = Cone(g_i \mid g_i \in I_2)$. Cone I is the cone generated by the generators of I_1 that are* subsumed *by I_2.*

Example 8. Returning to Example 7, we consider cone iterates I_4, I_5. In this case generator subsumption reduces to a simple containment check. Since generator $\mathbf{4} -$ **count** is not subsumed in I_5, we arrive at $I'_5 \equiv I_4 \widetilde{\nabla} I_5 = \widetilde{I^*} = I_\infty$.

Note 1. Alternatively, one can define dual widening as a widening operator [12,1] over the dual polyhedron that the generators of I_1, I_2 give rise to. On the set of PWL loop benchmarks our dual widening approach and those based on [12] and [1] produce identical fixed points where the difference in timings is not statistically significant.

Table 1. Summary of results: $|X|$ is the number of program variables; $|\mathcal{T}|$ - transitions; $\#$ - iterations to convergence; $\widetilde{\nabla}$ - use of dual widening. Lines (Rays) is the number of resultant inductive expectation equalities (inequalities). Time is taken on a MacBook Pro (2.4 GHz) laptop with 8 GB RAM, running MacOS X 10.9.1 (where $\varepsilon = 0.05$ sec).

Name	Description	$\lvert X\rvert$	$\lvert\mathcal{T}\rvert$	Iters		Fixpoint-gen		Time
				#	$\widetilde{\nabla}$	Lines	Rays	
MOT-EXAMPLE	Motivating Example of Figure 1	3	2	2	No	2	1	$\leq \varepsilon$
MOT-EX-LOOP-INV	Example 1 with added loop invariants	3	2	2	No	2	2	0.10
MOT-EX-POLY	Ex. 1 generate poly constr (deg ≤ 2)	9	2	2	No	5	2	0.18
2D-WALK	Random walk in 2 dimensions	4	4	7	Yes	3	1	$\leq \varepsilon$
AGGREGATE-RV	Accumulate RVs	3	2	2	No	2	0	$\leq \varepsilon$
HARE-TURTLE	Stochastic Hare-Turtle race	3	2	2	No	1	1	$\leq \varepsilon$
COUPON5	Coupon Collector's Problem (n = 5)	2	5	2	No	1	2	$\leq \varepsilon$
FAIR-COIN-BIASED	Simulating biased coin with fair coin	3	2	3	No	1	1	$\leq \varepsilon$
HAWK-DOVE-FAIR	Stochastic 2-player game (collaborate)	6	2	2	No	4	1	$\leq \varepsilon$
HAWK-DOVE-BIAS	Stochastic 2-player game (exploit)	6	2	2	No	3	1	$\leq \varepsilon$
FAULTY-INCR	Faulty incrementor	2	2	7	Yes	1	1	$\leq \varepsilon$

5 Experimental Results and Future Work

We present the experimental results of our prototype implementation that relies on PPL [1] for manipulating the polyhedral representations of cones. Table 1 presents the summary of the experiments we conducted on a set of probabilistic benchmarks. In [4], we present a description of these models and the expectation invariants obtained.

In all experiments we emphasize precision over computational effort. All examples except MOT-EX-LOOP-INV and MOT-EX-POLY run in under $\varepsilon = 0.05$ seconds, so we choose not to report these timing. Accordingly, dual widening $\widetilde{\nabla}$ delay was set sufficiently large at $K = 5$ to only force finite convergence but not to speed up computation. Nevertheless, the iterations converge quite fast and in many cases without the use of widening. Programs 2D-WALK and FAULTY-INCR require the widening ($\widetilde{\nabla}$) operator to ensure convergence. In all cases, *line* generators of the final pre-fixed point yield expectation invariants like $\mathbb{E}(\mathsf{e}) = 0$ and *rays* yield the invariants $\mathbb{E}(\mathsf{e}) \geq 0$.

Comparison with PRINSYS[11]. PRINSYS[11] implements the constraint-based quantitative invariant synthesis approach developed by Katoen et al. [13]. The tool uses a manually supplied template with unknown coefficients. The REDUCE computer algebra system is used to perform quantifier elimination and simplify the constraints. We applied PRINSYS with a linear template expression $\sum_j c_j x_j$ for state variables x_j in the program. Our comparison was carried out over 6 benchmark examples that are distributed with the tool. The comparison checked whether PRINSYS could discover quantitative invariants discovered by our approach. From a total set of 28 inductive expectation invariants our tool generates, PRINSYS could generate 3 of them. This shows that mutual inductive expectation invariants investigated in this paper are significant

for probabilistic loops. Next, we attempted to check whether PRINSYS can discover additional linear quantitative invariants not discovered by our approach due to the incompleteness of widening. Unfortunately, this check turned out inconclusive at the time of the experiment. The existing PRINSYS implementation automatically generates and simplifies nonlinear constraints on the template coefficients. However, the process of deriving an actual quantitative invariant requires manually extracting solutions from a set of nonlinear inequalities. Our manual efforts failed to find new invariants unique to the PRINSYS tool, but the overall comparison remains incomplete since we could not arguably find all solutions manually. However, it is important to observe that PRINSYS can generate invariants for templates that include indicator functions, while our technique does not. Similarly, PRINSYS handles nondeterminism in the programs, while we do not. The full details of the comparison can be found in the extended version [4].

Ongoing/Future Work. In many of the benchmark examples presented, invariants found using standard abstract interpretation by treating the stochastic choices as demonic nondeterminism help improve the quality of our expectation invariants. Going further, we would like to combine classical abstract interpretation with the techniques presented here to handle programs that mix non-deterministic and stochastic choices. Finally, we demonstrate polynomial invariant synthesis in Example MOT-EX-POLY by instrumenting monomials of fixed degree ($\deg \leq 2$) as fresh variables. Our analysis is thus able to generate *polynomial* expectation invariants such as $\mathbb{E}(4x^2-4xy+y^2 \mid n) \geq 0$, and $\mathbb{E}(4x^2-4xy+y^2-y+6 \mid n) = 0$. A sound formalization of polynomial invariant generation under relaxed independence conditions, and generalization of this approach to higher-order moments are also part of our future work.

Acknowledgments. The authors thank the anonymous reviewers for their insightful comments and Friedrich Gretz for helping us compare our work with PRINSYS. This work was supported by US National Science Foundation (NSF) under award number 1320069. All opinions are those of the authors and not necessarily of the NSF.

References

1. Antonopoulos, T., Gorogiannis, N., Haase, C., Kanovich, M., Ouaknine, J.: Foundations for decision problems in separation logic with general inductive predicates. In: Muscholl, A. (ed.) FOSSACS 2014. LNCS, vol. 8412, pp. 411–425. Springer, Heidelberg (2014)
2. Bouissou, O., Goubault, E., Goubault-Larrecq, J., Putot, S.: A generalization of p-boxes to affine arithmetic. Computing 94(2-4), 189–201 (2012)
3. Chakarov, A., Sankaranarayanan, S.: Probabilistic program analysis with martingales. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 511–526. Springer, Heidelberg (2013)
4. Chakarov, A., Sankaranarayanan, S.: Expectation invaraiants for probabilistic program loops as fixed points (2014) (extended version) (Draft, Available upon request)
5. Chung, K.L.: A course in probability theory, vol. 3. Academic Press, New York (1974)
6. Colón, M.A., Sankaranarayanan, S., Sipma, H.B.: Linear invariant generation using nonlinear constraint solving. In: Hunt Jr., W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 420–432. Springer, Heidelberg (2003)

7. Cousot, P., Cousot, R.: Abstract Interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: ACM Principles of Programming Languages, pp. 238–252 (1977)
8. Cousot, P., Halbwachs, N.: Automatic discovery of linear restraints among the variables of a program. In: POPL 1978, pp. 84–97 (January 1978)
9. Cousot, P., Monerau, M.: Probabilistic abstract interpretation. In: Seidl, H. (ed.) ESOP 2012. LNCS, vol. 7211, pp. 169–193. Springer, Heidelberg (2012)
10. Dubhashi, D., Panconesi, A.: Concentration of Measure for the Analysis of Randomized Algorithms. Cambridge University Press (2009)
11. Gretz, F., Katoen, J.-P., McIver, A.: Prinsys - on a quest for probabilistic loop invariants. In: Joshi, K., Siegle, M., Stoelinga, M., D'Argenio, P.R. (eds.) QEST 2013. LNCS, vol. 8054, pp. 193–208. Springer, Heidelberg (2013)
12. Halbwachs, N.: Détermination automatique de relations linéaires vérifiées par les variables d'un programme. PhD thesis, Institut National Polytechnique de Grenoble-INPG (1979)
13. Katoen, J.-P., McIver, A.K., Meinicke, L.A., Morgan, C.C.: Linear-invariant generation for probabilistic programs. In: Cousot, R., Martel, M. (eds.) SAS 2010. LNCS, vol. 6337, pp. 390–406. Springer, Heidelberg (2010)
14. Kozen, D.: Semantics of probabilistic programs. J. Comput. Syst. Sci. 22(3), 328–350 (1981)
15. Mardziel, P., Magill, S., Hicks, M., Srivatsa, M.: Dynamic enforcement of knowledge-based security policies. In: 2011 IEEE 24th Computer Security Foundations Symposium (CSF), pp. 114–128. IEEE (2011)
16. McAdams, H., Arkin, A.: It's a noisy business! genetic regulation at the nanomolar scale. Trends Genetics 15(2), 65–69 (1999)
17. McIver, A., Morgan, C.: Abstraction, Refinement and Proof for Probabilistic Systems. Monographs in Computer Science. Springer (2004)
18. Monniaux, D.: Abstract interpretation of probabilistic semantics. In: SAS 2000. LNCS, vol. 1824, pp. 322–340. Springer, Heidelberg (2000)
19. Monniaux, D.: Backwards abstract interpretation of probabilistic programs. In: Sands, D. (ed.) ESOP 2001. LNCS, vol. 2028, pp. 367–382. Springer, Heidelberg (2001)
20. Monniaux, D.: Abstract interpretation of programs as markov decision processes. Science of Computer Programming 58(1), 179–205 (2005)
21. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge University Press (1995)
22. Sankaranarayanan, S., Chakarov, A., Gulwani, S.: Static analysis for probabilistic programs: inferring whole program properties from finitely many paths. In: PLDI, pp. 447–458. ACM (2013)
23. Sankaranarayanan, S., Sipma, H.B., Manna, Z.: Constraint-based linear-relations analysis. In: Giacobazzi, R. (ed.) SAS 2004. LNCS, vol. 3148, pp. 53–68. Springer, Heidelberg (2004)
24. Williams, D.: Probability with Martingales. Cambridge University Press (1991)