Expectation Invariants for Probabilistic Program Loops as Fixed Points

Aleksandar Chakarov*[1] and Sriram Sankaranarayanan †[1]

[1]University of Colorado, Boulder, CO

October 9, 2014

Abstract

We present static analyses for probabilistic loops using *expectation invariants*. Probabilistic loops are imperative while-loops augmented with calls to random value 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 Random` and `Normal` (`Gaussian`). Probabilistic programs arise in a variety of domains ranging from biological systems [18] to randomized algorithms [23]. 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, 20, 22, 9, 17] 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 [19]. First we extend quantitative invariants from single expressions to a set of expressions that are mutually invariant: multiple expressions 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.

*first.lastname@colorado.edu

†first.lastname@colorado.edu

Our approach first defines the notion of inductive invariants using the pre-expectation operator, along the lines of McIver and Morgan [19]. 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 [24]. 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* [19]. 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, 19]. 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 [26]. 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, 25], implemented in the tool PRINSYS [11]. An experimental comparison is provided in Section 5.

Abstract domains for probabilistic programs were first considered by Monniaux [20], by enriching standard abstract domains with bounds on the measure. Refinements of this idea appear in the work of Mardziel et al [17] 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 [21]. 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

```
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
```

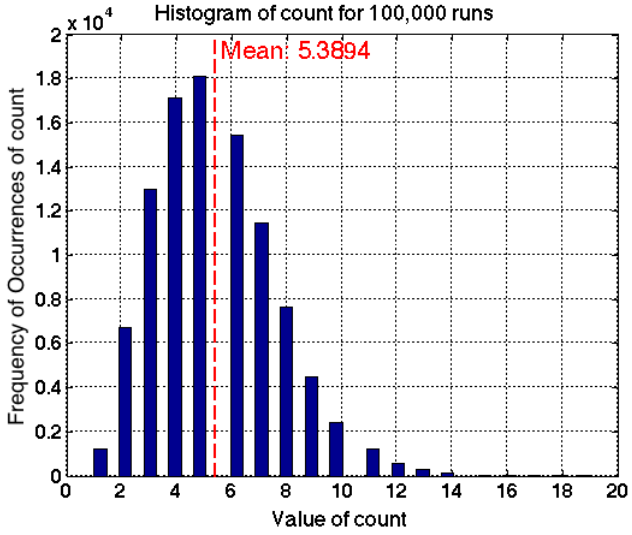

Figure 1: (**Left, Top**) Simple example of a probabilistic program loop; (**Left, Bottom**) Modified loop with *stuttering* semantics, and (**Right**) histogram of the value of count after executing the stuttering loop for at most 25 steps.

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 1.1. 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(**Left, Bottom**)). 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 $\mathbb{E}(\text{count})$ 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 $\mathbb{E}(\text{count})$. Figure 1(**Right**) presents the simulation results in the form of a frequency histogram that estimate the expected value of the loop counter.

In this paper we propose a static analysis approach, instead, 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:

$$\forall\, n \in \mathbb{N},\ \mathbb{E}(3\text{count} - 2\text{y} + 2 \mid n) = 0 \ \text{ and } \mathbb{E}(4\text{x} + 4\text{y} - 9\text{count} \mid n) = 0.$$

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.

2 Preliminaries

2.1 Probabilistic Loops

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

Definition 2.1 (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* $\mathcal{F}_i(\mathbf{x}, \mathbf{r})$ that yields the next state $\mathbf{x}' := \mathcal{F}_i(\mathbf{x}, \mathbf{r})$.

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 we reserve as future work.

Definition 2.2 (PWL Transitions)**.** A *piecewise linear* transition $\tau : \langle \mathbf{g}, \mathcal{F}(\mathbf{x}, \mathbf{r}) \rangle$ has the following special structure:

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

$$F(\mathbf{x}, \mathbf{r}) = \begin{cases} \mathsf{f_1} : A_1\mathbf{x} + B_1\mathbf{r}_c + d_1, & \text{with probability } p_1, \\ \vdots & \\ \mathsf{f_k} : A_k\mathbf{x} + B_k\mathbf{r}_c + d_k, & \text{with probability } p_k, \end{cases}$$

  Options $\mathsf{f_1}, \ldots, \mathsf{f_k}$, abstract the effect of the Bernoulli choices in $\mathbf{r}_b$, and are called *forks*, while $p_1, \ldots, p_k$ are *fork probabilities* satisfying $0 < p_i \leq 1$, and $\sum_{i=1}^{k} p_i = 1$; with $A_1, \ldots, A_k \in \mathbb{R}^{m \times m}$, $B_1, \ldots, B_k \in \mathbb{R}^{m \times l}$, 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 of transitions $\tau_1 : \langle \mathbf{g}_1, \mathcal{F}_1 \rangle$ and $\tau_2 : \langle \mathbf{g}_2, \mathcal{F}_2 \rangle$ in $\mathcal{T}$, $\mathbf{g}_1 \wedge \mathbf{g}_2 \equiv \mathit{false}$.

Exhaustiveness: For all transitions τ_i, we require that $\bigvee_{\tau_i \in \mathcal{T}} \mathbf{g}_i \equiv \mathit{true}$.

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 $\mathbf{x}$.

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

Definition 2.3 (Sample Path)**.** A *sample path* (or an execution) of the loop is an infinite sequence $(\mathbf{x}_0, 0) \xrightarrow{\tau^{(0)}, \mathbf{r}_0} (\mathbf{x}_1, 1) \xrightarrow{\tau^{(1)}, \mathbf{r}_1} (\mathbf{x}_2, 2) \to \cdots \xrightarrow{\tau^{(n)}, \mathbf{r}_n} (\mathbf{x}_{n+1}, n+1) \to \cdots$, wherein, **(a)** $(\mathbf{x}_0, 0)$ is a sample from $\mathcal{D}_0$ and **(b)** for each $i \geq 0$, $(\mathbf{x}_{i+1}, i+1)$ is obtained by executing the unique transition $\tau^{(i)} : (\mathbf{g}_i, \mathcal{F}_i)$ that is enabled on the state $(\mathbf{x}_i, i)$. 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 $\mathbf{r}_c$ to obtain $\mathbf{x}_{i+1} = \mathcal{F}_i(\mathbf{x}_i, \mathbf{r}_i)$.

We demonstrate the definitions above on a simple example.

Example 2.1. In Figure 1 (**Left, Bottom**) 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 stuttering 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:

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

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 $\mathrm{x}, \mathrm{y}, \mathrm{count}$. 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.

2.2 Operator Semantics

Probabilistic program semantics can be thought of as continuous linear operators 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 [15, 21].

We now describe the construction of the distribution $\mathcal{D}_{n+1}$ for the $(n+1)^{th}$ loop iteration, given the distribution $\mathcal{D}_n$ at the n^{th} step. *The details of this section may be skipped upon a first reading.*

Formally, $\mathcal{D}_n$ consists of a probability space $\langle \Omega_n, \mathcal{F}_n, \mu_n \rangle$ over a sample set Ω_n with events in $\mathcal{F}_n$ and a probability measure μ_n, as well as a Borel measurable function $D_n : \Omega_n \to \mathbb{R}^m$ mapping samples $\omega \in \Omega_n$

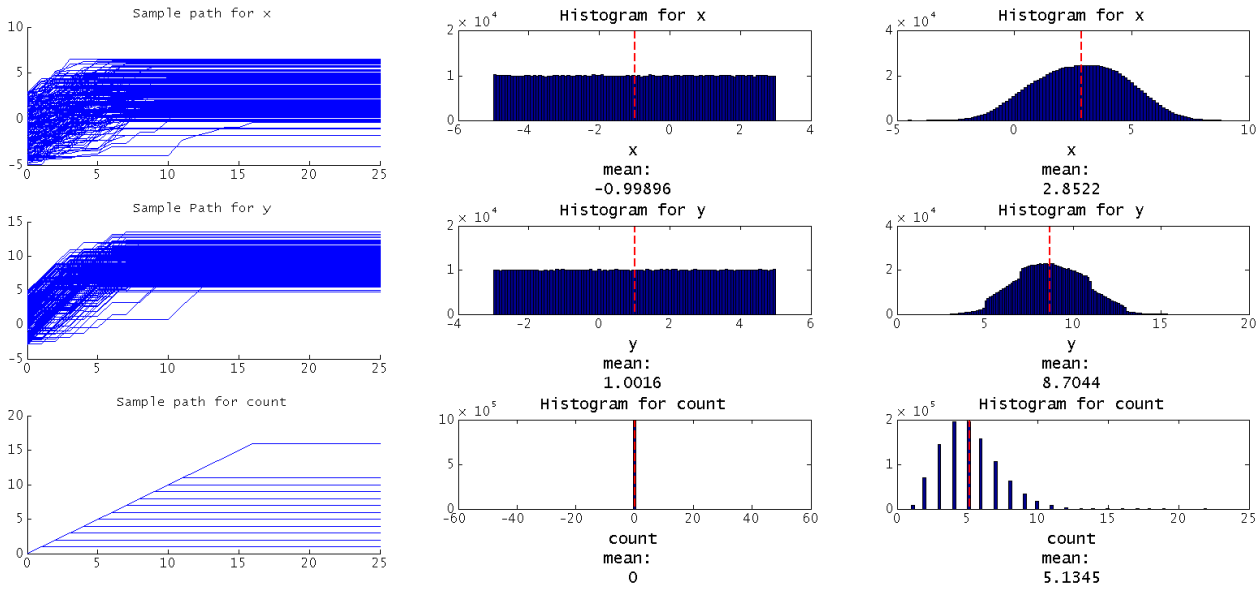

Figure 2: (**Left**) Some sample paths for the program in Figure 1. (**Right**) Frequency histograms for the distributions $\mathcal{D}_n$ for n = 0, 25.

to values of the state variable $\mathbf{x}$. Likewise, the distribution of random $\mathbf{r}$ is given by $\mathcal{D}_R$, defined as the probability space $\langle \Omega_r, \mathcal{F}_r, \mu_r \rangle$ and a measurable function $D_r : \Omega_r \to \mathbb{R}^k$ mapping samples $\omega_r \in \Omega_r$ to random variables $\mathbf{r}$.

The standard product measure $\mathcal{D}_{n,R} : \mathcal{D}_n \otimes \mathcal{D}_R$ has sample set $\Omega_{n+1} : \Omega_n \times \Omega_r$ and events $\mathcal{F}_{n,R}$ generated by sets of the form $A \times B$ where $A \in \mathcal{F}_n$ and $B \in \mathcal{F}_r$ with a measurable function $D_{n,r} : \Omega_n \times \Omega_r \to \mathbb{R}^m \times \mathbb{R}^k$, mapping events (ω_n, ω_r) to corresponding values of state and random variables $(\mathbf{x} : D_n(\omega_n),\ \mathbf{r} : D_r(\omega_r))$.

The distribution $\mathcal{D}_{n+1}$ is defined naturally by the product probability space $\mathcal{D}_{n,R}$ with a measurable function $D_{n+1} : \Omega_n \times \Omega_r \mapsto \mathbb{R}^m$ maps a sample (ω_n, ω_r) from the sample space $\Omega_n \times \Omega_r$ to the state $\mathbf{x}'$ that results from the execution of the probabilistic loop for a single step starting from the state $\mathbf{x} : D_n(\omega_n)$ and random values $\mathbf{r} : D_r(\omega_r)$.

$$D_{n+1}(\omega_n, \omega_r) : \{\mathbf{x}' \mid \mathbf{x} = D_n(\omega_n),\ \mathbf{r} = D_r(\omega_r),\ \mathbf{x} \xrightarrow{\mathbf{r}} \mathbf{x}'\}.$$

The absence of nondeterminism in our setup ensures that $\mathbf{x}'$ is uniquely defined given samples ω_n, and ω_r. To summarize, the next state distribution $\mathcal{D}_{n+1}$ has a probability space defined as the product probability space of $\mathcal{D}_n \otimes \mathcal{D}_r$ with a measurable function D_{n+1} that maps samples from $\mathcal{D}_n$ and $\mathcal{D}_r$ to the resulting states obtained by a single step execution of the loop.

Sample Path Semantics: Equivalently, a direct characterization of the distribution $\mathcal{D}_n$ is given as the product of the probability spaces

$$\mathcal{D}_0 \otimes \underbrace{\mathcal{D}_R \otimes \cdots \otimes \mathcal{D}_R}_{n \text{ times}},$$

equipped with a measurable function $F_n : \Omega_0 \times \Omega_r^n \mapsto \mathbb{R}^m$ that maps each sample $\hat{\omega} : (\omega_0, \omega_{r1}, \ldots, \omega_{rn})$ to the unique resulting state $\mathbf{x}_n$ obtained by starting from initial state $\mathbf{x}_0 : D_0(\omega_0)$ and executing the loop n times with a series of random values $\mathbf{r}_1 : D_r(\omega_{r1}),\ \ldots\ \mathbf{r}_n : D_r(\omega_{rn})$, at each iteration to yield a sample path:

$$\mathbf{x}_0 : D_0(\omega_0) \xrightarrow{\mathbf{r}_1 :\ D_r(\omega_{r1})} \mathbf{x}_1 \xrightarrow{\mathbf{r}_2 :\ D_r(\omega_{r2})} \cdots \xrightarrow{\mathbf{r}_n :\ D_r(\omega_{rn})} \mathbf{x}_n$$

The absence of nondeterminism ensures that the sample path is unique once $\mathbf{x}_0, \mathbf{r}_1, \ldots, \mathbf{r}_n$ are known.

Existence of Moments We assume that the distributions $\mathcal{D}_0$ and $\mathcal{D}_R$ are independent, and all moments exist (and are finite). Therefore, for any polynomial $p(\mathbf{x}_0, \mathbf{r}_1, \ldots, \mathbf{r}_m)$ wherein $\mathbf{x}_0 \sim \mathcal{D}_0$ and $\mathbf{r}_i \sim \mathcal{D}_R$, with variables $\mathbf{r}_i, \mathbf{r}_j$ pairwise independent for $i \neq j$, $\mathbb{E}_{\mathcal{D}_0 \otimes \mathcal{D}_R \cdots \otimes \mathcal{D}_R}(p)$ exists (and is finite).

Let $\mathcal{P}$ be a piecewise linear probabilistic program and $\mathtt{e}(\mathbf{x}_n)$ be a polynomial expression over the variables $\mathbf{x}_n$ obtained after n steps of loop execution. Under the assumptions above on only $\mathcal{D}_0$ and $\mathcal{D}_r$, we conclude the following key result:

Lemma 2.1. *For any polynomial expression* $\mathtt{e}(\mathbf{x})$ *over the program variables, and any* $n \in \mathbb{N}$, $\mathbb{E}_{\mathcal{D}_n}(\mathtt{e})$ *exists (and is finite).*

2.3 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 [19]. Let $\tau : \langle \mathbf{g}, \mathcal{F} \rangle$ be a transition and $\mathtt{e}[\mathbf{x}]$ be an expression involving the state variables $\mathbf{x}$ of the program.

Given any state $\mathbf{x}$ of the loop and an expression $\mathtt{e}$ involving the state variables, we seek to know the expected value of the expression $\mathtt{e}(\mathbf{x}')$ evaluated over all possible next states $\mathbf{x}'$ that can be reached in one loop iteration starting from $\mathbf{x}$. The resulting function that maps $\mathbf{x}$ to the expectation of the $\mathtt{e}$ in the next step is called the *pre-expectation* of $\mathtt{e}$, following the terminology established earlier by McIver and Morgan [19]. We first defined the pre-expectation w.r.t a given transition τ and then use this to define the pre-expectation w.r.t to all transitions in the loop.

The pre-expectation operator $\mathtt{pre}\mathbb{E}_\tau$ of an expression $\mathtt{e}$ involving the next-state program variables X', w.r.t to a transition τ is an *expression transformer* that computes the expectation of the expression $\mathbf{e}(\mathbf{x}')$ in the next step in terms of *current state variables of the program* $\mathbf{x}$. Formally,

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

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

Notation: We define the pre-expectation as a map that transforms expressions over the *next-state variables* X' into current state variable expression X. Therefore, wherever applicable, $\mathtt{pre}\mathbb{E}(\mathtt{e}(\mathbf{x}'))$ results in an expression over the original state variables $\mathbf{x}$. For convenience, let $\mathtt{e}'$ denote the expression $\mathtt{e}[\mathbf{x} \mapsto \mathbf{x}']$ with each current state variable x_j substituted by its next-state variable x'_j.

Consider a PWL transition τ with $k > 0$ forks, $\mathtt{f_1}, \ldots, \mathtt{f_k}$, each of the form $\mathtt{f_j} : A_j\mathbf{x} + B_j\mathbf{r} + d_j$ with fork probability p_j. The pre-expectation operator is defined as

$$\mathtt{pre}\mathbb{E}_\tau(\mathtt{e}') = \sum_{j=1}^{k} p_j \mathbb{E}_R(\textsc{pre}(\mathtt{e}', f_j) \mid \mathbf{x})$$

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

Example 2.2. 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.

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

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

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

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

wherein $\mathbb{1}_g(\mathbf{x})$ is the *indicator* function:

$$\mathbb{1}_g(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} \models g(\mathbf{x}), \\ 0 & \text{otherwise} \end{cases}$$

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

Lemma 2.2. *The expected value of* $\texttt{e}$ *over the post-state distribution starting from state* $(\mathbf{x}_n, n)$ *is the value of the pre-expectation* $\hat{\texttt{e}}$ *evaluated over the current state* $\mathbf{x}_n$*:*

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

Proof. This follows directly from the definition of the pre-expectation operator. □

Finally, we extend Lemma 2.2 to the full *distribution* $\mathcal{D}_n$ from which $\mathbf{x}_n$ is drawn.

Lemma 2.3. *Let* $\texttt{e}$ *be an affine program expression. Then*

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

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 the notion of expectation invariants and derive a fixed point characterization of expectation invariants in the next section.

3.1 Definitions and Examples

Let $\mathcal{P} : \langle \mathcal{T}, \mathcal{D}_0, n \rangle$ be a probabilistic loop and let $\langle \mathbf{x}_0, 0 \rangle$ be the initial state of the system. From Section 2 we know that $\mathbf{x}_0$ 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 $\texttt{e}$ *at time step* n as $\mathbb{E}(\texttt{e} \mid n) = \mathbb{E}_{\mathcal{D}_n}(\texttt{e})$.
Notation: We denote the expectation of an expression $\texttt{e}$ over the program variables at the j^{th} step as $\mathbb{E}(\texttt{e} \mid n = j)$ or equivalently $\mathbb{E}_{\mathcal{D}_j}(\texttt{e})$. Unless otherwise mentioned, $\texttt{e}$ will denote an affine (or linear) expression over the program variables.

Definition 3.1 (Expectation Invariants)**.** An $\texttt{e}$ over the program variables X is called an *expectation invariant* (EI) iff for all $n \geq 0$, $\mathbb{E}(\texttt{e} \mid n) \geq 0$.

Thus, expectation invariants are program expressions whose expectations over the initial distribution are non-negative, and under *any* number $n \geq 0$ of iterations of the probabilistic loop remain non-negative.

Example 3.1. 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.

3.2 Martingales and Expectation Invariants

Expectation invariants as given by Definition 3.1 are closely related to the concept of (super-, sub-) *martingales*, studied in our earlier work [3].

Definition 3.2 (Martingales)**.** Let $s : \langle \mathbf{x}_n, n \rangle$ be the state of a probabilistic loop $\mathcal{P}$. Let e be a program expression over the program variables X. Expression e is called:

- a *supermartingale* if $\forall\ n, \mathbf{x}$, $\texttt{pre}\mathbb{E}(\texttt{e}') \leq \texttt{e}$,
- a *submartingale* if $\forall\ n, \mathbf{x}$, $\texttt{pre}\mathbb{E}(\texttt{e}') \geq \texttt{e}$,
- a *martingale* if e is both a supermartingale and a submartingale.

In fact, martingales naturally yield expectation invariants.

Lemma 3.1. *For every supermartingale expression* e *the expression* $e_0 - \texttt{e}$ *is an expectation invariant, wherein* $e_0 = \mathbb{E}(\texttt{e} \mid n = 0)$.

Proof. First, we observe that $\forall\ \mathbf{x}$, $\texttt{pre}\mathbb{E}(\texttt{e}') \leq \texttt{e}$. Therefore, assuming that the expectations on both sides exist, we have for all $n \geq 0$, $\mathbb{E}_{\mathcal{D}_n}(\texttt{pre}\mathbb{E}(\texttt{e}')) \leq \mathbb{E}_{\mathcal{D}_n}(\texttt{e})$.

We now prove that $\mathbb{E}(e_0 - \texttt{e} \mid n) \geq 0$ for all $n \in \mathbb{N}$ by induction. Clearly, for $n = 0$, the statement holds. Furthermore, assume that $\mathbb{E}(e_0 - \texttt{e} \mid n = j)$ holds, we obtain

$$\begin{aligned}\mathbb{E}(e_0 - \texttt{e} \mid n = j+1) &= \mathbb{E}(e_0 - \texttt{pre}\mathbb{E}(\texttt{e}') \mid n = j) \\ &= e_0 - \mathbb{E}(\texttt{pre}\mathbb{E}(\texttt{e}') \mid n = j) \\ &\geq e_0 - \mathbb{E}(\texttt{e} \mid n = j) \geq 0.\end{aligned}$$

□

However, expectation invariants can arise without martingales, as shown by the following simple example that repeatedly swaps two variables x,y:

```
real x := rand(0, 5), y := rand(4, 7)
while (true)
    (x, y) := (y + unifRand(-1,1), x+unifRand(-2,2))
```

Notice that expressions x and y are expectation invariant. However, they are not martingales. In fact, to prove that $\mathbb{E}(\texttt{x}) \geq 0$, at any step, we require that $\mathbb{E}(\texttt{y}) \geq 0$ at the previous step.

Therefore, the notion of expectation invariants subsumes that of martingales as defined here. Drawing analogies to the familiar case of Floyd-Hoare invariants, martingales correspond to assertions which are invariant by themselves, whereas expectation invariants are analogous to the general case of *mutually inductive invariants* [16].

3.2.1 Proving Expectation Invariance

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 3.3 (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 $p(\mathbf{x})$ over the program variables, $\mathbb{E}_{\mathcal{D}}(p(\mathbf{x}))$ exists, and is finite.

[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.

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

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, we invoke Lemma 2.1 to conclude that $\mathcal{D}_n$ is admissible for each $n \geq 0$. However, rather than construct $\mathcal{D}_n$ explicitly for each n (which can be impractical), we formulate the principle of inductive expectation invariants. Consider expressions $E = \{\mathsf{e}_1, \ldots, \mathsf{e}_m\}$ wherein each e_i is a linear (or polynomial) expression involving the program variables.

Definition 3.4 (Inductive Expectation Invariants)**.** The set E of expressions forms an inductive expectation invariant of the program $\mathcal{P}$ iff for each e_j, $j \in [1, m]$,

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

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

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 $\mathcal{D}$.

Theorem 3.1. *Let $E : \{\mathsf{e}_1, \ldots, \mathsf{e}_m\}$ be inductive expectation invariants (Definition 3.4), it follows that each $\mathsf{e}_j \in E$ is an expectation invariant of the program:*

$$\forall\ n \in \mathbb{N},\ \mathbb{E}(\mathsf{e}_j|n) \geq 0$$

Proof. The proof uses the important fact that each distribution $\mathcal{D}_n$ is admissible. We prove by simultaneous induction that

$$\bigwedge_{j=1}^{m} \mathbb{E}(\mathsf{e}_j|n) \geq 0\,.$$

Base-Case: The base case for $n = 0$ follows from item 1 of Definition 3.4.
Induction Step: Let us assume that the required statement holds for n and attempt to show for $n+1$. Using Eq. (1), and the admissibility of $\mathcal{D}_n$, we note that for each $j \in [1, m]$,

$$(\mathbb{E}_{\mathcal{D}_n}(\mathsf{e}_1) \geq 0 \ \wedge \ \cdots \ \wedge \ \mathbb{E}_{\mathcal{D}_n}(\mathsf{e}_m) \geq 0) \ \models \ \mathbb{E}_{\mathcal{D}_n}(\mathtt{pre}\mathbb{E}(\mathsf{e}'_j)) \geq 0$$

Therefore, we conclude that $\mathbb{E}_{\mathcal{D}_n}(\mathtt{pre}\mathbb{E}(\mathsf{e}'_j)) \geq 0$. Since, $\mathbb{E}_{\mathcal{D}_{n+1}}(\mathsf{e}_j) = \mathbb{E}_{\mathcal{D}_n}(\mathtt{pre}\mathbb{E}(\mathsf{e}'_j))$, we have $\mathbb{E}_{\mathcal{D}_{n+1}}(\mathsf{e}_j) = \mathbb{E}_{\mathcal{D}_n}(\mathtt{pre}\mathbb{E}(\mathsf{e}'_j)) \geq 0$ for each j. Thus, the induction step is proven. □

However, Definition 3.4 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 3.2 (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) $\mathsf{e}, \mathsf{e}_1, \ldots, \mathsf{e}_k$:*

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

3. $\mathbb{E}_{\mathcal{D}}(\mathbb{1}_{\varphi}\mathsf{e} + \mathbb{1}_{\neg\varphi}\mathsf{e}) = \mathbb{E}_{\mathcal{D}}(\mathsf{e})$*, provided* $[\![\varphi]\!]$ *is measurable.*

Using these facts as "axioms", we attempt to reformulate the key step 2 of Definition 3.4 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 $\mathtt{pre}\mathbb{E}(\mathsf{e}_j)$ for each $\mathsf{e}_j \in E$ as

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

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 $E = \{\mathsf{e}_1, \ldots, \mathsf{e}_m\}$.

Lemma 3.2. *Suppose for all* $\mathsf{e}_i \in E$*, the principle* (2) *holds:*

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

for some $\lambda_{j,i} \geq 0, \mu_{j,p} \geq 0$ *and* $\varphi_p \models g_p \geq 0$*, then* E *satisfies the original induction principle* (1)*:*

$$\underline{\textit{For all admissible } \mathcal{D},}\ (\mathbb{E}_{\mathcal{D}}(\mathsf{e}_1) \geq 0\ \wedge\ \cdots\ \wedge\ \mathbb{E}_{\mathcal{D}}(\mathsf{e}_m) \geq 0)\ \models\ \mathbb{E}_{\mathcal{D}}(\mathtt{pre}\mathbb{E}(\mathsf{e}'_j)) \geq 0\,.$$

Proof. Let E be such that for each $\mathsf{e}_i \in E$, we satisfy (2) as below:

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

for some $\lambda_{j,i} \geq 0, \mu_{j,p} \geq 0$ and $\varphi_p \models g_p \geq 0$.

Let $\mathcal{D}$ be any admissible distribution such that $\bigwedge_{j=1}^{m} \mathbb{E}_{\mathcal{D}}(\mathsf{e}_j) \geq 0$. Using, linearity of expectation, we note that

$$\mathbb{E}_{\mathcal{D}}\left(\sum_{i=1}^{m} \lambda_{j,i}\mathsf{e}_i\right) = \sum_{i=1}^{m} \lambda_{j,i} \underbrace{\mathbb{E}_{\mathcal{D}}(\mathsf{e}_i)}_{\geq 0} \geq 0\,. \tag{3}$$

Similarly, applying Theorem 3.2, we note that $\mathbb{E}_{\mathcal{D}}(\mathbb{1}_{\varphi_p} \times g_p) \geq 0$.

$$\mathbb{E}_{\mathcal{D}}\left(\sum_{p} \mu_{j,p}\ (\mathbb{1}_{\varphi_p} \times g_p)\right) = \sum_{p} \mu_{j,p} \underbrace{\mathbb{E}_{\mathcal{D}}\left(\mathbb{1}_{\varphi_p} \times g_p\right)}_{\geq 0} \geq 0 \tag{4}$$

Combining, (3) and (4), we note that

$$\begin{array}{rcll} \mathbb{E}_{\mathcal{D}}(\mathtt{pre}\mathbb{E}(\mathsf{e}'_j)) & = & \mathbb{E}_{\mathcal{D}}\left(\sum_{i=1}^{m} \lambda_{j,i}\mathsf{e}_i + \sum_{p} \mu_{j,p}\ (\mathbb{1}_{\varphi_p} \times g_p)\right) & \text{From Stmt. of Thm.} \\ & = & \mathbb{E}_{\mathcal{D}}\left(\sum_{i=1}^{m} \lambda_{j,i}\mathsf{e}_i\right) + \mathbb{E}_{\mathcal{D}}\left(\sum_{p} \mu_{j,p}\ (\mathbb{1}_{\varphi_p} \times g_p)\right) & \\ & \geq & 0 & \text{Applying (3) and (4)} \end{array}$$

□

3.3 Conic Inductive Expectation Invariants

We now formalize this intuitive notion of inductive invariants 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 $\mathbf{g_i} : (g_{i,1} \ \ldots \ g_{i,n_i})^T$ 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 $\mathbf{e} : \ (\mathtt{e}_1, \ldots, \mathtt{e}_m)^T$.

Definition 3.5 (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: $\mathbb{E}_{\mathcal{D}_0}(\mathtt{e}_j) \geq 0$ over the initial distribution $\mathcal{D}_0$;
2. Induction Step: There exists a vector of multipliers $\lambda_j \geq 0$, 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 $\mathbf{g}_l$:

$$\underline{\text{For each } \mathtt{e}_j} \ (\exists \ \lambda_j \geq 0) \ (\forall \ \tau_l \in \mathcal{T}) \ (\exists \ \mu_l \geq 0) \ \mathtt{pre}\mathbb{E}_{\tau_l}(\mathtt{e}_j) = \lambda_j^T \ \mathbf{e} + \mu_l^T \mathbf{g}_l \, . \tag{5}$$

In particular, we note that the order of quantification in Equation (5) is quite important. We note for a given expression $\mathtt{e}_j$ the multipliers λ_j must stay the same across all the transitions $\tau_l \in \mathcal{T}$. This will ensure the applicability of the linearity of expectation.

Example 3.2. 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 1.1. 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 $\lambda : (1 \ 0 \ 0 \ 0 \ 0)^T$, we obtain $\mathtt{pre}\mathbb{E}(\mathtt{e}_1) : \ \lambda^T \mathbf{e} + \mathbf{0} \times \mathbb{1}_{x+y \leq 10}$.

Changing the order of quantification in Equation 5 makes the rule unsound. In particular, we will address the need to maintain the multipliers λ_j the same across all transitions. Consider a variant of the Equation (5), as below:

$$\underline{\text{For each } \mathtt{e}_j} \ (\forall \ \tau_l \in \mathcal{T}) \ (\exists \ \boldsymbol{\lambda_j} \geq \mathbf{0}) \ (\exists \ \mu_l \geq 0) \ \mathtt{pre}\mathbb{E}_{\tau_l}(\mathtt{e}'_j) = \lambda_j^T \ \mathbf{e} + \mu_l^T \mathbf{g}_l \, . \tag{6}$$

Such a rule *seems* like a natural encoding of the implication:

$$\bigwedge_{j=1}^{m} \mathtt{e}_j \geq 0 \ \wedge \ \bigwedge_{k=1}^{q} g_{l,k} \geq 0 \ \models \ \mathtt{pre}\mathbb{E}_{\tau_l}(\mathtt{e}'_j) \geq 0 \, .$$

The following example demonstrates the *unsoundness* of the rule (6).

Example 3.3. Consider the program below:

```
real x := unifRand(-1,1)
while (true)
      if ( x <= 0)
         x := 2 * x;
      else
         x := x / 2;
```

$$\begin{array}{rcl} X & : & \{\mathrm{x}\} \\ \mathcal{T} & : & \{\tau_1, \tau_2\} \\ \tau_1 & : & \begin{cases} \mathbf{g}_1 : \ x < 0 \\ F_1(x) : \ 2x \end{cases} \\ \tau_2 & : & \begin{cases} \mathbf{g}_2 : \ x \geq 0 \\ F_2(x) : \ 0.5x \end{cases} \\ \mathcal{D}_0 & : & \mathrm{Uniform}[-1, 1] \end{array}$$

First, we observe that $\mathbb{E}_{\mathcal{D}_0}(\mathrm{x}) = 0$. This gives us two IEI candidates x and -x. Using the rule in Equation 6 we obtain:

- For transition τ_1, we have

$$\mathtt{pre}\mathbb{E}_{\tau_1}(\mathrm{x}') = 2 \times (\mathrm{x}), \quad \text{and} \quad \mathtt{pre}\mathbb{E}_{\tau_1}(\text{-}\mathrm{x}') = 2 \times (\text{-}\mathrm{x});$$

- For transition τ_2, we have

$$\mathtt{pre}\mathbb{E}_{\tau_2}(\mathrm{x}') = 0.5 \times (\mathrm{x}), \quad \text{and} \quad \mathtt{pre}\mathbb{E}_{\tau_2}(\text{-}\mathrm{x}') = 0.5 \times (\text{-}\mathrm{x}).$$

This means that according to rule (6), we can conclude that $\mathbb{E}(\mathrm{x} \mid n) \geq 0$ and $\mathbb{E}(\text{-}\mathrm{x} \mid n) \geq 0$, and so $\mathbb{E}(\mathrm{x} \mid n) = 0$ for all $n \geq 0$. This is clearly false since any negative initial value of x only ever execute τ_1 and grows unbounded!

The correct version of the rule (5), is able to correctly prove the invariance of -x and disprove x.

Lemma 3.3. *Let $E : \{\mathtt{e}_1, \ldots, \mathtt{e}_m\}$ be a conic inductive invariant for a program $\mathcal{P}$ as given by Definition 3.5. It follows that each $\mathtt{e}_j$ satisfies Equation* (2)*:*

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

for $\lambda_{j,i}, \mu_{i,p} \geq 0$ and $\varphi_p \models g_p \geq 0$.

Proof. We note that for each $\mathtt{e}_i$, the expression:

$$\mathtt{pre}\mathbb{E}(\mathtt{e}_i') : \sum_{\tau \in \mathcal{T}} \mathbb{1}_{g_\tau} \times \mathtt{pre}\mathbb{E}_\tau(\mathtt{e}_i')\,. \tag{7}$$

From (2), there exists λ such that for each transition τ,

$$\mathtt{pre}\mathbb{E}_\tau(\mathtt{e}_i') = \lambda^T \mathbf{e} + \mu^T \mathbf{g}_\tau\,.$$

Here the guard assertion for τ is given by $\mathbf{g}_\tau \geq 0$. Substituting this into Equation (7) yields,

$$\begin{array}{rcl} \mathtt{pre}\mathbb{E}(\mathtt{e}_i') & = & \sum_{\tau \in \mathcal{T}} \mathbb{1}_{\mathbf{g}_\tau \geq 0} \times \left(\lambda^T \mathbf{e} + \mu_\tau^T \mathbf{g}_\tau\right) \\ & = & \sum_{\tau \in \mathcal{T}} \mathbb{1}_{\mathbf{g}_\tau \geq 0} \times \left(\lambda^T \mathbf{e}\right) + \sum_{\tau \in \mathcal{T}} \mathbb{1}_{\mathbf{g}_\tau \geq 0} \times \left(\mu_\tau^T \mathbf{g}_\tau\right) \\ & = & \lambda^T \sum_{\tau \in \mathcal{T}} \mathbb{1}_{\mathbf{g}_\tau \geq 0} \times \mathbf{e} + \sum_{\tau \in \mathcal{T}} \mathbb{1}_{\mathbf{g}_\tau \geq 0} \times \left(\mu_\tau^T \mathbf{g}_\tau\right). \end{array}$$

In particular, we note that having a common set of multipliers λ across transitions allows us to rewrite the summation $\sum_{\tau \in \mathcal{T}} \mathbb{1}_{\mathbf{g}_\tau \geq 0} \times \left(\lambda^T \mathbf{e}\right)$ as $\lambda^T \sum_{\tau \in \mathcal{T}} \mathbb{1}_{\mathbf{g}_\tau \geq 0} \times \mathbf{e}$. Next, since the transition guards are mutually exclusive and exhaustive, it follows that $\sum_{\tau \in \mathcal{T}} \mathbb{1}_{\mathbf{g}_\tau \geq 0} \times \mathbf{e} = \mathbf{e}$. Therefore, we write

$$\mathtt{pre}\mathbb{E}(\mathtt{e}_i') = \lambda^T \mathbf{e} + \sum_{\tau \in \mathcal{T}} \mathbb{1}_{\mathbf{g}_\tau \geq 0} \times \mu_\tau^T \mathbf{g}_\tau\,.$$

This concludes the proof.

□

Theorem 3.3. *Let E be a conic inductive invariant for a program $\mathcal{P}$ as given by Definition 3.5. It follows that each $\mathtt{e}_j \in E$ is an expectation invariant of the program.*

Proof. Proof simply combines Lemma 3.3 with Lemma 3.2. □

3.4 Pre-Expectation Closed Cones

Thus far, we have presented inductive expectation invariants as a finite set of expressions $E = \{\mathsf{e}_1, \ldots, \mathsf{e}_m\}$, satisfying the conditions in Definition 3.4 or 3.5. We transfer our notion from a finite set of expressions to a finitely generated cone of these in preparation for our fixed point characterization in the next section.

Definition 3.6 (Cones)**.** Let $E = \{\mathsf{e}_1, \ldots, \mathsf{e}_k\}$ be a finite set of program expressions over the program variables $\mathbf{x}$. 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 3.4. Revisiting Example 3.2, 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,

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

Conic Representations: A finitely generated cone $\mathsf{Cone}(\mathsf{e}_1, \ldots, \mathsf{e}_k)$ of affine expressions $\mathsf{e}_1, \ldots, \mathsf{e}_k$ is represented using a polyhedral cone. Specifically, let $\mathsf{e} : \; c_0 + \mathbf{c}^T\mathbf{x}$ be any element of the cone. The polyhedral cone representation uses variables $(c_0, \mathbf{c})$. Such a polyhedron can be represented using the constraint representation as:

$$C : \; P \begin{pmatrix} \mathbf{c} \\ c_0 \end{pmatrix} \leq 0$$

or as a set of generators given by the coefficient vectors of the expressions $\mathsf{e}_1, \ldots, \mathsf{e}_k$.

Example 3.5. Consider the cone generated by expressions

$$\mathsf{e}_1 : \; -2x + y - 3, \; \mathsf{e}_2 : \; 4x - 3z + 4\,.$$

Any element of the cone can be written as $c_0 + c_1 x + c_2 y + c_3 z$ wherein the constraints:

$$(\exists\, \lambda_1, \lambda_2 \geq 0) \begin{bmatrix} c_0 = -3\lambda_1 + 4\lambda_2 \;\wedge\; c_1 = -2\lambda_1 \\ c_2 = \lambda_1 \;\wedge\; c_3 = -3\lambda_2 \end{bmatrix}$$

Alternatively, we may express the cone with a vertex $(c_0, c_1, c_2, c_3) : \; (0, 0, 0, 0)$ and rays:

$$\left\{ \begin{pmatrix} -3 \\ -2 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 4 \\ 4 \\ 0 \\ -3 \end{pmatrix} \right\}$$

4 Expectation Invariants as Fixed Points

In this section, we show that the notion of conic invariants as presented in Definition 3.5 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 $\mathbf{x}$ 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 $c_0 + \mathbf{c}^T\mathbf{x}$ for $c_0 \in \mathbb{R}, \mathbf{c} \in \mathbb{R}^n$. Let $\mathbb{A}(\mathbf{x})$ represent the set of all affine expressions over $\mathbf{x}$.

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 $\mathbf{x}$. 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 3.5.

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

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 $E_{\mathcal{D}_0}(\mathbf{x})$ 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 $x_i = \mathbb{E}_{\mathcal{D}_0}(x_i)$. The representation of the initial cone is given by the set of $2n$ rays of the form $[\mathbb{E}_{\mathcal{D}_0}(x_i)\ 0\ \cdots\ 0\ \pm 1\ 0\ \cdots\ 0]$.

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

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

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 λ that shows its membership back in the cone. This can be seen as a way to ensure the proper order of quantification in Definition 3.5.

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 $(c_0, \mathbf{c})$ representing the expression coefficients and λ for the multipliers.

Lemma 4.1. *For a given cone C, the pre-expectation operator across a transition $\mathtt{pre}\mathbb{E}_\tau(C)$ is also a cone.*

PreExpectation of Cones: First, we define the lifting of $\mathtt{pre}\mathbb{E}_\tau(I)$ for a single cone of expressions I. Let τ be given by the guard set $\bigwedge_{i=1}^{l} \mathbf{g}_i^T\mathbf{x} + h_i \geq 0$, and update with forks $f_1, \ldots, f_k$ wherein $f_i : A_i\mathbf{x} + B_i\mathbf{r} + \mathbf{a}_i$

is taken with probability p_i. Consider a generic next state affine expression: $\mathtt{e}: \ c_0 + \mathbf{c}^T\mathbf{x}'$. We write

$$\mathtt{pre}\mathbb{E}_\tau(\mathtt{e}'): \ c_0 + \mathbf{c}^T(p_1A_1 + \cdots + p_kA_k)\mathbf{x} + \mathbf{c}^T\mathbb{E}_{\mathcal{D}_R}(p_1B_1\mathbf{r} + \cdots + p_kB_k\mathbf{r}) + \mathbf{c}^T(p_1\mathbf{a}_1 + \cdots + p_k\mathbf{a}_k)\,.$$

Simplifying, we write

$$\mathtt{pre}\mathbb{E}_\tau(\mathtt{e}'): \ (c_0 + \alpha^T\mathbf{c}) + \mathbf{c}^T\mathcal{B}\mathbf{x}\,.$$

Let I be the cone generated by the expressions $\{\mathtt{e}_1, \ldots, \mathtt{e}_k\}$ wherein $\mathtt{e}_j: \ d_{0,j} + \mathbf{d}_j^T\mathbf{x}$. The generators of the cone are given by the rays $(d_{0,j}, \mathbf{d}_j)$ for $j = 1, \ldots, k$. We compute an augmented representation of the cone I given by the constraints:

$$P_I(d_0, \mathbf{d}, \lambda, \mu): \ \lambda \geq 0 \ \wedge \ \mu \geq 0 \ \wedge \ d_0 = \sum_{j=1}^{k} \lambda_j d_{0,j} + \sum_{i=1}^{l} \mu_i h_i \ \wedge \ \mathbf{d} = \sum_{j=1}^{k} \lambda_j \mathbf{d}_j + \sum_{i=1}^{l} \mu_i \mathbf{g}_i\,.$$

The polyhedron P_I represents all combinations of expressions $(d_0 + \mathbf{d}^T\mathbf{x})$ that are derived by a conic combination of expressions $\mathtt{e}_1, \ldots, \mathtt{e}_k$ through multipliers $\lambda_1, \ldots, \lambda_k \geq 0$ and guard inequality expressions $\mathbf{g}_1, \ldots, \mathbf{g}_l$ through multipliers $\mu_1, \ldots, \mu_l \geq 0$. The cone $\mathtt{pre}\mathbb{E}_\tau(I)$ is given as

$$\mathtt{pre}\mathbb{E}_\tau(I): \ (\exists\ \mu)\ P_I(c_0 + \alpha^T\mathbf{c}, \mathcal{B}^T\mathbf{c}, \lambda, \mu)\,.$$

Note that $\mathtt{pre}\mathbb{E}_\tau(I)$ is a polyhedron over variables $(c_0, \mathbf{c})$ representing an expression $\mathtt{e}: \ c_0 + \mathbf{c}^T\mathbf{x}$ and multipliers $\lambda \in \mathbb{R}^k$.

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

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

An expression $\mathtt{e}$ belongs to $\mathtt{pre}\mathbb{E}(I)$ if for some $\lambda \geq 0$, $(e, \lambda) \in \mathtt{pre}\mathbb{E}_{\tau_j}(I)$ 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 $C(I'): \ (\exists\ \lambda) \bigcap_{j=1}^{k} C(\hat{I}_j)$, representing $I': \mathtt{pre}\mathbb{E}(I)$, by intersecting the cones $C(\hat{I}_j)$ and projecting the dimensions corresponding to λ.

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 4.1. *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 (Definition 3.5) iff $I: \mathsf{Cone}(E)$ is a pre-fixed point of $\mathcal{G}$, i.e, $I \subseteq \mathcal{G}(I)$.*

4.1 Proof of Theorem 4.1

The details of the proof are quite intricate, so we build the proof of Theorem 4.1 in a bottom up fashion.

Let $\mathcal{P}: \langle \mathcal{T}, \mathcal{D}_0, n\rangle$ be a probabilistic loop with $\mathcal{T} = \{\tau_1, \ldots, \tau_m\}$, where each transition is of the form $\tau_i: \langle \mathbf{g}_i, \mathcal{F}_i\rangle$. We begin by showing that $\mathtt{pre}\mathbb{E}_\tau$ is a monotone operator.

Lemma 4.2. *Let $I_1 = Cone(e_1, \ldots, e_m)$ and $I_2 = Cone(h_1, \ldots, h_k)$ be two finitely generated cones such that $I_1 \subseteq I_2$ and let $\tau_i \in \mathcal{T}$, then $\mathtt{pre}\mathbb{E}_{\tau_i}(I_1) \subseteq \mathtt{pre}\mathbb{E}_{\tau_i}(I_2)$.*

Proof. Let $(\mathsf{e}, \lambda) \in \mathtt{pre}\mathbb{E}_{\tau_i}(I_1)$ for some $\mathsf{e} \in \mathbb{A}(x)$ and $\lambda \geq 0$. By Definition 4, there exists $\mu_i \geq 0$ such that $\mathtt{pre}\mathbb{E}_{\tau_i}(\mathsf{e}) = \lambda^T(e_1 \cdots e_j)^T + \mu_i^T \mathbf{g}_i$.

Since $I_1 \subseteq I_2$ then for every generator e_j of I_1 there exist non-negative coefficients $\lambda_{j1}, \ldots, \lambda_{jk}$ such that $e_j = \lambda_j^T(h_1 \cdots h_k)$. Therefore, we can define the *change of basis transformation* matrix $\Lambda = \begin{bmatrix} \lambda_{11} & \cdots & \lambda_{1k} \\ \vdots & \ddots & \vdots \\ \lambda_{m1} & \cdots & \lambda_{mk} \end{bmatrix}$ such that $\begin{bmatrix} e_1 \\ \vdots \\ e_m \end{bmatrix} = \Lambda \begin{bmatrix} h_1 \\ \vdots \\ h_k \end{bmatrix}$. Notice that Λ is independent of τ_i; moreover, Λ is a non-negative matrix.

This means that $\mathtt{pre}\mathbb{E}_{\tau_i}(\mathsf{e}) = \lambda^T(e_1 \cdots e_m)^T + \mu_i^T \mathbf{g}_i = \lambda^T \Lambda (h_1 \cdots h_k)^T + \mu_i^T \mathbf{g}_i$. Therefore, there exists a non-negative $\lambda' \equiv \lambda^T \Lambda$ such that (e, λ') belongs to I_2. □

A consequence of Lemma 4.2 we see that for every (e, λ) pair in I_1, there exists a unique (e, λ') pair in I_2 regardless of the guard $\mathbf{g}_i$ (and therefore, transition τ_i). The following result follows immediately.

Lemma 4.3 (Monotonicity of $\mathtt{pre}\mathbb{E}$)**.** *Let $I_1 = Cone(e_1, \ldots, e_m)$ and $I_2 = Cone(h_1, \ldots, h_k)$ such that $I_1 \subseteq I_2$, then $\mathtt{pre}\mathbb{E}(I_1) \subseteq \mathtt{pre}\mathbb{E}(I_2)$.*

Proof. If $\mathbf{e} \in \mathtt{pre}\mathbb{E}(I_1)$ then there exists a signature $\lambda \geq 0$ such that $(\mathsf{e}, \lambda) \in \bigcap_{\tau \in \mathcal{T}} \mathtt{pre}\mathbb{E}_\tau(I_1)$. Define $\lambda' \equiv \lambda^T \Lambda$. Then (e, λ') belongs to $\bigcap_{\tau \in \mathcal{T}} \mathtt{pre}\mathbb{E}_\tau(I_2)$. Therefore, e belongs to $\mathtt{pre}\mathbb{E}(I_2)$. This completes the proof of monotonicity. □

Lemma 4.4. *Let E be a conic inductive invariant, then $Cone(E) \subseteq I_0$.*

Proof. WLOG, let $\mathsf{e}_i \in E$, then by construction, e_i is a generator of $Cone(E)$. By Definition 3.5, we know that $\mathbb{E}_{\mathcal{D}_0}(\mathsf{e}_i) = k$, for some $k \geq 0$. On the other hand, e_i is a linear expression, so $\mathsf{e}_i : c_0 + \mathbf{c}^T \mathbf{x}$ for some $c_0 \geq 0$, $\mathbf{c} \geq 0$.

$$I_0 \equiv Cone(\{1, x_1 - \mathbb{E}_{\mathcal{D}_0}(x_1), \mathbb{E}_{\mathcal{D}_0}(x_1) - x_1, \cdots, x_n - \mathbb{E}_{\mathcal{D}_0}(x_n), \mathbb{E}_{\mathcal{D}_0}(x_n) - x_n\}).$$

For every $j \geq 1$, define $(\lambda_{2j-1}, \lambda_{2j}) \equiv (c_j, 0)$ if $c_j \geq 0$ and $(0, -c_j)$ otherwise. Finally, define $\lambda_0 \equiv k - \sum c_j$. It then follows that $\mathsf{e}_i = \lambda^T \mathbf{i_0}$ with $\lambda = (\lambda_0, \lambda_1, \cdots, \lambda_{2n}) \geq 0$.

Therefore, $\mathsf{e}_i \in I_0$. □

Lemma 4.5. *Let E be a conic inductive invariant and $I = Cone(E)$, then $I \subseteq \mathtt{pre}\mathbb{E}(I)$.*

Proof. Let $\mathsf{e} \in I$, then by Definition 3.5, there exists a certificate $\lambda \geq 0$ that satisfies the requirements of Definition 4, simultaneously, for every transition.

Therefore, $\mathsf{e} \in \mathtt{pre}\mathbb{E}(I)$. □

Theorem 4.2 (Theorem 4.1)**.** *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. 3.5) if and only if $I : \mathsf{Cone}(E)$ is a pre-fixed point of $\mathcal{G}$, i.e, $I \subseteq \mathcal{G}(I)$.*

Proof. **Part 1:** Let $I_1 = Cone(e_1, \ldots, e_m)$ and $I_2 = Cone(h_1, \ldots, h_k)$ such that $I_1 \subseteq I_2$. Expanding the definition of $\mathcal{G}$ and applying Lemma 4.3, $\mathcal{G}(I_1) = I_0 \cap \mathtt{pre}\mathbb{E}(I_1) \subseteq I_0 \cap \mathtt{pre}\mathbb{E}(I_2) = \mathcal{G}(I_2)$. This completes the proof that if $I_1 \subseteq I_2$ then $\mathcal{G}(I_1) \subseteq \mathcal{G}(I_2)$.

Part 2 ($\Rightarrow$): Let E be a conic inductive expectation invariant, then by Lemma 4.4, $I = Cone(E)$ is a subset of I_0. By Lemma 4.5, we know $I \subseteq \mathtt{pre}\mathbb{E}(I)$. Therefore, by definition of $\mathcal{G}$, I is a pre-fixed point of $\mathcal{G}$.

($\Leftarrow$): Let $I = Cone(E)$ for some set of expressions E such that $I \subseteq \mathcal{G}(I)$. Then $I \subseteq I_0 \cap \mathtt{pre}\mathbb{E}(I)$. Since $I \subseteq \mathtt{pre}\mathbb{E}(I)$ then there exists a certificate $\lambda \geq 0$ common for all transitions τ; this satisfies condition (2) of

Definition 3.5. Now let $\mathsf{e} : c_0 + \mathbf{c}^T\mathbf{x}$ be a linear expression in I. Since $I \subseteq I_0$, then $\mathsf{e} : \lambda'_0 + \sum_{i=1}^{m}[\lambda'_{2i-1}(x_i - \mathbb{E}_{\mathcal{D}_0}(x_i)) + \lambda'_{2i}(\mathbb{E}_{\mathcal{D}_0}(x_i) - x_i)] = \lambda'_0 + \sum_{i=1}^{m}[(\lambda'_{2i-1} - \lambda'_{2i})x_i + (\lambda'_{2i} - \lambda'_{2i-1})\mathbb{E}_{\mathcal{D}_0}(x_i)]$ for some non-negative scalars λ'_j. Define $\kappa_i \equiv \lambda'_{2i-1} - \lambda'_{2i}$. The expectation $\mathbb{E}_{\mathcal{D}_0}(\mathsf{e}) = \mathbb{E}_{\mathcal{D}_0}(\lambda'_0 + \sum_{i=1}^{m}[\kappa_i x_i - \kappa_i \mathbb{E}_{\mathcal{D}_0}(x_i)]) \geq 0$. Therefore, e satisfies Definition 3.5(1). Thus, E is a conic inductive expectation invariant. □

4.2 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 3.5. 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 4.2 (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 4.1. 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, **4** $-$ **count**, count, x, $-$x, y, $-$y, dist, $-$dist	$c_0 + 4c_4 \geq 0$, $c_0 \geq 0$
I_1	1, **1** $-$ **count**, count, x, $-$x, y, $-$y, dist, $-$dist	$c_0 + c_4 \geq 0$, $c_0 \geq 0$	I_5	1, **5** $-$ **count**, count, x, $-$x, y, $-$y, dist, $-$dist	$c_0 + 4c_4 \geq 0$, $c_0 \geq 0$
I_2	1, **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, **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 4.3 (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 4.2. Returning to Example 4.1, we consider cone iterates I_4, I_5. In this case generator subsumption reduces to a simple containment check. Since generator **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	$\|X\|$	$\|\mathcal{T}\|$	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 1.1 with added loop invariants	3	2	2	No	2	2	0.10
MOT-EX-POLY	Ex. 1.1 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 $\mathbb{E}(\mathtt{e}) = 0$ and *rays* yield the invariants $\mathbb{E}(\mathtt{e}) \geq 0$.

5.0.1 Comparison with PRINSYS[11].

PRINSYS[11] implements the constraint-based quantitative invariant synthesis approach developed by Katoen et al. [14]. 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 all state variables x_j in the program. Our comparison was carried out over the 6 benchmark examples distributed with the tool. The comparison checked whether PRINSYS could discover quantitative invariants discovered by our approach. Table 2 presents a summary of the comparison.

From a total set of 26 inductive expectation invariants our tool generates, PRINSYS could generate 3 of them. Notice that we had to manually provide some additional initial invariants which PRINSYS was able to trivially, yet correctly prove invariant (denoted by asterisk in last column). Overall, we observe that mutual inductive expectation invariants investigated in this paper provide interesting, significant facts about the probabilistic loops in the PRINSYS benchmarks.

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

Table 2: Summary of comparison results: IEI - invariants generated by our tool; Iters, Time - number of iterations, time for our tool to converge; PRINSYS - was PRINSYS able to infer this quantitative invariant.

Name	IEI	Iters	Time	PRINSYS
BIASED-COIN	2 - 2b - count = 0 3 - 4x $\geq$ 0	6	0.0877	No No
BINOMIAL-UPDATE (M=20)	4x - 3n = 0 x $\geq$ 0	21	0.1337	No No
COWBOYS	1 - 7turn - continue $\geq$ 0 6 - 2turn - 6continue - 5count = 0	4	0.1146	No No
FAIR-COIN	x - y = 0 3 - 4x $\geq$ 0 -4x + 3count $\geq$ 0	6	0.0844	Yes No No
GEOMETRIC	x $\geq$ 0 3x - flip = 0 4x - count = 0 count $\geq$ 0	29	0.1988	No No No Yes*
UNLIMITED-MARTINGALE	rounds $\geq$ 0 32c + rounds $\leq$ 1600 c + b = 51 (10 additional inequalities)	13	0.2	Yes* No No No

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.

Finally, it is important to observe that PRINSYS can generate invariants for templates that include indicator functions, while our technique curently does not. Similarly, PRINSYS handles nondeterminism in the programs, while we do not.

Ongoing/Future Work. In many of the benchmark examples we present, we found that invariants discovered 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 nondeterministic 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] R. Bagnara, P. M. Hill, E. Ricci, and E. Zaffanella. Precise widening operators for convex polyhedra. In *Static Analysis*, pages 337–354. Springer, 2003.

[2] O. Bouissou, E. Goubault, J. Goubault-Larrecq, and S. Putot. A generalization of p-boxes to affine arithmetic. *Computing*, 94(2-4):189–201, 2012.

[3] A. Chakarov and S. Sankaranarayanan. Probabilistic program analysis with martingales. In *CAV*, pages 511–526, 2013.

[4] A. Chakarov and S. Sankaranarayanan. Expectation invaraiants for probabilistic program loops as fixed points (extended version), 2014. Draft, Available upon request.

[5] K. L. Chung. *A course in probability theory*, volume 3. Academic press New York, 1974.

[6] M. Colón, S. Sankaranarayanan, and H. Sipma. Linear invariant generation using non-linear constraint solving. In *CAV*, volume 2725 of *LNCS*, pages 420–433. Springer, July 2003.

[7] P. Cousot and R. Cousot. Abstract Interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *ACM Principles of Programming Languages*, pages 238–252, 1977.

[8] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among the variables of a program. In *POPL'78*, pages 84–97, Jan. 1978.

[9] P. Cousot and M. Monerau. Probabilistic abstract interpretation. In H. Seidel, editor, *22nd European Symposium on Programming (ESOP 2012)*, volume 7211 of *Lecture Notes in Computer Science*, pages 166–190, Heidelberg, 2012. Springer-Verlag.

[10] D. Dubhashi and A. Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, 2009.

[11] F. Gretz, J.-P. Katoen, and A. McIver. Prinsys - on a quest for probabilistic loop invariants. In *QEST*, pages 193–208, 2013.

[12] N. Halbwachs. *Détermination automatique de relations linéaires vérifiées par les variables d'un programme*. PhD thesis, Institut National Polytechnique de Grenoble-INPG, 1979.

[13] J.-P. Katoen, A. McIver, L. Meinicke, and C. Morgan. Linear-invariant generation for probabilistic programs. In *Static Analysis Symposium (SAS)*, volume 6337 of *LNCS*, page 390406. Springer, 2010.

[14] J.-P. Katoen, A. K. McIver, L. A. Meinicke, and C. C. Morgan. Linear-invariant generation for probabilistic programs. In *Static Analysis*, pages 390–406. Springer, 2011.

[15] D. Kozen. Semantics of probabilistic programs. *J. Comput. Syst. Sci.*, 22(3):328–350, 1981.

[16] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety*. Springer, New York, 1995.

[17] P. Mardziel, S. Magill, M. Hicks, and M. Srivatsa. Dynamic enforcement of knowledge-based security policies. In *Computer Security Foundations Symposium (CSF), 2011 IEEE 24th*, pages 114–128. IEEE, 2011.

[18] H. McAdams and A. Arkin. It's a noisy business! genetic regulation at the nanomolar scale. *Trends Genetics*, 15(2):65–69, 1999.

[19] A. McIver and C. Morgan. *Abstraction, Refinement And Proof For Probabilistic Systems (Monographs in Computer Science)*. SpringerVerlag, 2004.

[20] D. Monniaux. Abstract interpretation of probabilistic semantics. In *Static Analysis Symposium*, volume 1824 of *Lecture Notes in Computer Science*, pages 322–339. Springer, 2000.

[21] D. Monniaux. Backwards abstract interpretation of probabilistic programs. In *Programming Languages and Systems*, pages 367–382. Springer, 2001.

[22] D. Monniaux. Abstract interpretation of programs as markov decision processes. *Science of Computer Programming*, 58(1):179–205, 2005.

[23] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[24] S. Sankaranarayanan, A. Chakarov, and S. Gulwani. Static analysis for probabilistic programs: inferring whole program properties from finitely many paths. In *PLDI*, pages 447–458. ACM, 2013.

[25] S. Sankaranarayanan, H. B. Sipma, and Z. Manna. Constraint-based linear-relations analysis. In *Static Analysis Symposium (SAS 2004)*, volume 3148 of *LNCS*, pages 53–69. Springer, August 2004.

[26] D. Williams. *Probability with Martingales*. Cambridge University Press, 1991.

6 Probabilistic Loop Benchmarks

6.1 Example of Fixpoint Computation for PWL Loops

In this section we present the application of the expectation invariant analysis to the concrete program in Example 1.1. We start with the largest possible cone of linear program expressions whose initial expectation $\mathbb{E}(\mathbf{e} \mid 0) \geq 0$. Our goal will be to refine this cone of candidate expressions into a cone of expectation invariant expressions.

Initially, we know the following facts: $\mathbb{E}(\mathrm{x} \mid 0) = -1$, $\mathbb{E}(\mathrm{y} \mid 0) = 1$, $\mathbb{E}(\mathit{count} \mid 0) = 0$. The set of expectation invariant candidate expressions (in PPL format) is then $E = \{1, \mathrm{x}+1, -\mathrm{x}-1, \mathrm{y}-1, 1-\mathrm{y}, \mathrm{count}, -\mathrm{count}\}$ since $\mathbb{E}(\mathtt{e} \mid 0) \geq 0$ hold for all $\mathtt{e} \in E$. Therefore, $I_0 \equiv \mathit{Cone}(E)$ is a finitely generated cone of program expressions with non-negative expectations at step 0.

Next, consider the pre-expectation of $\mathbf{e} : c_0 + c_1 x_1 + \cdots + c_m x_m$ under both transitions:

$$\begin{aligned}
\mathtt{pre}\mathbb{E}_{\tau_1}(\mathbf{e}) &= c_0 + c_1(\mathrm{x} + 0.75) + c_2(\mathrm{y} + 1.5) + c_3(\mathrm{count} + 1) \\
&= c_0 + 0.75c_1 + 1.5c_2 + c_3 + c_1\mathrm{x} + c_2\mathrm{y} + c_3\mathrm{count} \\
\mathtt{pre}\mathbb{E}_{\tau_2}(\mathbf{e}) &= c_0 + c_1\mathrm{x} + c_2\mathrm{y} + c_3\mathrm{count}.
\end{aligned}$$

Now $\mathcal{G}(I_0) = I_0 \cap \mathtt{pre}\mathbb{E}(I_0)$:

$$(\exists \overline{\lambda}^T \geq 0)(\forall \tau_i \in \mathcal{T})(\exists \mu_i \geq 0)\, \varphi(\mathbf{c}) : \mathtt{pre}\mathbb{E}_{\tau_i}(\mathbf{e}) = \overline{\lambda}^T \mathbf{E} + \mu_i^T \mathbf{g_i}.$$

Constraints $\varphi_0(\mathbf{c})$ are called dual (in the linear programming sense) because they constrain the coefficients of the program expressions.

	$\mathtt{pre}\mathbb{E}_{\tau_1}(\mathbf{e})$:		$\mathtt{pre}\mathbb{E}_{\tau_2}(\mathbf{e})$:
(1)	$c_0 + 0.75c_1 + 1.5c_2 + c_3 = \lambda_0 + \lambda_1 - \lambda_2 - \lambda_3 + \lambda_4 + 10\mu_1$	(5)	$c_0 = \lambda_0 + \lambda_1 - \lambda_2 - \lambda_3 + \lambda_4 - 10\mu_2$
(2)	$c_1 = -\lambda_1 + \lambda_2 - \mu_1$	(6)	$c_1 = -\lambda_1 + \lambda_2 + \mu_2$
(3)	$c_2 = \lambda_3 - \lambda_4 - \mu_1$	(7)	$c_2 = \lambda_3 - \lambda_4 + \mu_2$
(4)	$c_3 = \lambda_5 - \lambda_6$	(8)	$c_3 = \lambda_5 - \lambda_6$

These constraints are clearly satisfiable (even trivially satisfiable) and after performing projection we arrive at the following set of constraints: $\varphi_0(\mathbf{c}) : 6c_0 - 9c_1 - 4c_3 \geq 0 \wedge 3c_1 + 6c_2 + 4c_3 = 0$. The cone described by $\varphi_0(\mathbf{c})$ is precisely $\mathtt{pre}\mathbb{E}(I_0)$. Conjoining $\varphi_0(c)$ to the constraints of I_0, therefore, amounts to computing $I_1 \equiv \mathcal{G}(I_0) = I_0 \cap \mathtt{pre}\mathbb{E}(I_0)$. In other words, conjoining the constraints removes from the cone I_0 those expressions whose single step pre-expectation (even across one transition) does not belong to I_0.

The process is repeated where $\mathbf{E}$ is the generators of I_1. The reader is welcome to verify that $I_2 \equiv \mathcal{G}(I_1) = I_0 \cap \mathtt{pre}\mathbb{E}(I_1) = I_1$. Therefore, $I^* = I_2$ is the fixpoint.

Cone	Generators	Constraints
I_0	$1,\ \mathrm{x}+1,\ -\mathrm{x}-1,\ \mathrm{y}-1,$ $1-\mathrm{y},\ \mathrm{count},\ -\mathrm{count}$	$c_0 - c_1 + c_2 \geq 0$
I_1	$1,\ 2-2\mathrm{y}+3\mathrm{count},\ 4+4\mathrm{x}-3\mathrm{count},$ $-2+2\mathrm{y}-3\mathrm{count},\ -4-4\mathrm{x}+3\mathrm{count}$	$3c_1 + 6c_2 + 4c_3 = 0,\ c_0 - c_1 + c_2 \geq 0,$ $6c_0 - 9c_1 - 4c_3 \geq 0$
I_2	$1,\ 2-2\mathrm{y}+3\mathrm{count},\ 4+4\mathrm{x}-3\mathrm{count},$ $-2+2\mathrm{y}-3\mathrm{count},\ -4-4\mathrm{x}+3\mathrm{count}$	$3c_1 + 6c_2 + 4c_3 = 0,\ c_0 - c_1 + c_2 \geq 0,$ $6c_0 - 9c_1 - 4c_3 \geq 0$

Finally, confirm that every expression in I^* is expectation invariant by observing that $2-2\mathrm{y}+3\mathrm{count}$ and $4+4\mathrm{x}-3\mathrm{count}$ are martingales (Cf. Section 3.2). Therefore, every conic combination of generators in I^* is also expectation invariant.

6.2 mot-ex-loop-inv and mot-ex-poly

Example 6.1. Consider the program in Example 3.2. Figure 3 presents the same program annotated with addition loop invariants φ_{l_inv} obtained by running a traditional abstract interpretation analysis and

```
real x := rand(-5,3)
real y := rand(-3,5)
int count := 0
@loop_invariants {
     -5 <= x; -3 <= y <= 17;
      0 <= count;
      x + y <= 14;
      2*count - y + 5 >= 0;
      y - x + 6 >= 0;
      2*count - x + 3 >= 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
```

$$
\begin{aligned}
\mathbf{g}_1 \quad &: \quad (x+y\le 10)\wedge(-5\le x)\wedge(-3\le y\le 17)\\
&\quad \wedge(0\le \mathit{count})\wedge x+y\le 14\wedge(y-x+6\ge 0)\\
&\quad \wedge(2\mathit{count}-y+5\ge 0)\\
&\quad \wedge(2\mathit{count}-x+3\ge 0)\\
\mathcal{F}_{\tau_1} \quad &: \quad \begin{cases} f_1: \begin{bmatrix} \text{x'} & \mapsto & \text{x}+r_1, \\ \text{y'} & \mapsto & \text{y}+2, \\ \text{count'} & \mapsto & \text{count}+1, \end{bmatrix} & \text{w.p. } \frac{3}{4} \\ f_2: \begin{bmatrix} \text{x'} & \mapsto & \text{x}, \\ \text{y'} & \mapsto & \text{y}, \\ \text{count'} & \mapsto & \text{count}+1, \end{bmatrix} & \text{w.p. } \frac{1}{4} \end{cases}\\
\mathbf{g}_2 \quad &: \quad (x+y> 10)\wedge(-5\le x)\wedge(-3\le y\le 17)\\
&\quad \wedge(0\le \mathit{count})\wedge x+y\le 14\wedge(y-x+6\ge 0)\\
&\quad \wedge(2\mathit{count}-y+5\ge 0)\\
&\quad \wedge(2\mathit{count}-x+3\ge 0)\\
\mathcal{F}_{\tau_2} \quad &: \quad \begin{cases} \text{x'} & \mapsto & \text{x}, \\ \text{y'} & \mapsto & \text{y}, \\ \text{count'} & \mapsto & \text{count}, \end{cases}
\end{aligned}
$$

Figure 3: (**Left**) Modified version of the program in Example 1.1 by adding loop invariant annotations. These are produced by traditional (non-probabilistic) abstract interpretation; (**Right**) corresponding PWL probabilistic loop where loop invariants are added to the guards of transitions.

treating fork probabilities as non-deterministic choice. This additional information allowed us to prove a slightly richer set $E = \{\text{x}+1, 2\text{x}-\text{y}+3, 4\text{x}-3\text{count}+4, -2\text{x}+\text{y}-3, -4\text{x}+3\text{count}-4\}$ to be a conic inductive expectation invariant.

Consider $\mathtt{e}_1 : \text{x}+1$, then $\mathtt{pre}\mathbb{E}_{\tau_1}(\mathtt{e}_1) = \mathbb{E}_{r_1}\left(\frac{3}{4}(\text{x}+r_1+1)+\frac{1}{4}(\text{x}+1)\right) = \text{x}+\frac{7}{4}$. Since τ_2 is the *stuttering* transition it is easy to observe $\mathtt{pre}\mathbb{E}_{\tau_2}(\mathtt{e}_1) = \mathtt{e}_1$. In the fashion of Definition 3.5 and Example 3.2, we derive the following (dual) constraints:

	$\mathtt{pre}\mathbb{E}_{\tau_1}(\mathtt{e}_1)$:	$\mathtt{pre}\mathbb{E}_{\tau_2}(\mathtt{e}_1)$:
const	$\frac{7}{4} = \lambda_1 + 3\lambda_2 + 4\lambda_3 - 3\lambda_4 - 4\lambda_5 + 10\mu_{1,1}$ $+5\mu_{1,2} + 3\mu_{1,3} + 17\mu_{1,4} +$ $14\mu_{1,6} + 6\mu_{1,7} + 5\mu_{1,8} + 3\mu_{1,9}$	$1 = \lambda_1 + 3\lambda_2 + 4\lambda_3 - 3\lambda_4 - 4\lambda_5 - 10\mu_{2,1}$ $+5\mu_{2,2} + 3\mu_{2,3} + 17\mu_{2,4} +$ $14\mu_{2,6} + 6\mu_{2,7} + 5\mu_{2,8} + 3\mu_{2,9}$
x	$1 = \lambda_1 + 2\lambda_2 - 4\lambda_3 - 2\lambda_4 - 4\lambda_5 - \mu_{1,1}$ $+\mu_{1,2} - \mu_{1,6} - \mu_{1,7} - \mu_{1,9}$	$1 = \lambda_1 + 2\lambda_2 - 4\lambda_3 - 2\lambda_4 - 4\lambda_5 + \mu_{2,1}$ $+\mu_{2,2} - \mu_{2,6} - \mu_{2,7} - \mu_{2,9}$
y	$0 = -\lambda_2 + \lambda_4 - \mu_{1,1}$ $+\mu_{1,3} - \mu_{1,4} - \mu_{1,6} + \mu_{1,7} - \mu_{1,8}$	$0 = -\lambda_2 + \lambda_4 + \mu_{2,1}$ $+\mu_{2,3} - \mu_{2,4} - \mu_{2,6} + \mu_{2,7} - \mu_{2,8}$
count	$0 = -3\lambda_3 + 3\lambda_5$ $+\mu_{1,5} + 2\mu_{1,8} + 2\mu_{1,9}$	$0 = -3\lambda_3 + 3\lambda_5$ $+\mu_{2,5} + 2\mu_{2,8} + 2\mu_{2,9}$

The careful reader is welcome to verify the following solution: set all coefficients to 0 except $\lambda_1 = \frac{50}{80}$, $\lambda_2 = \frac{15}{80}$, $\mu_{1,3} = \frac{15}{80}$, $\mu_{2,1} = \frac{6}{80}$, $\mu_{2,3} = \frac{3}{80}$, $\mu_{2,7} = \frac{6}{80}$. As can be seen from this solution, the addition of the loop invariants to the guards makes them more "expressive" in the sense of the (λ, μ)-decomposition.

Note 2. *The program in Figure 3 is identical to the one in Figure 1(**Middle**) except for the added loop invariant annotations. While these do not change the behavior of the program, they improve the results of our analysis.*

Variety	Fixed Point	
	Generators	Constraints
MOT-EX	$\mathrm{y} - 2\mathrm{x},\ 3 + 2\mathrm{x} - \mathrm{y},\ -3 - 2\mathrm{x} + \mathrm{y},$ $4\mathrm{x} + 4\mathrm{y} - 9\mathrm{count},\ -4\mathrm{x} - 4\mathrm{y} + 9\mathrm{count}$	$3c_1 + 6c_2 + 4c_3 = 0$ $c_0 - c_1 + c_2 \geq 0$
MOT-EX-LOOP-INV	$\mathrm{count},\ 56 - 9\mathrm{count},$ $-2 + 2\mathrm{y} - 3\mathrm{count},\ 2 - 2\mathrm{y} + 3\mathrm{count}$ $-4 - 4\mathrm{x} + 3\mathrm{count},\ 4 + 4\mathrm{x} - 3\mathrm{count}$	$9c_0 + 33c_1 + 93c_2 + 56c_3 \geq 0,$ $c_0 - c_1 + c_2 \geq 0$
MOT-EX-POLY	$4\mathrm{x}^2 - 4\mathrm{xy} + \mathrm{y}^2,\ 207\mathrm{x} + 89\mathrm{x}^2 - 98\mathrm{xy} + 20\mathrm{y}^2,$ $36\mathrm{x} - 27\mathrm{count} + 16\mathrm{x}^2 - 16\mathrm{xy} + 4\mathrm{y}^2,$ $-36\mathrm{x} + 27\mathrm{count} - 16\mathrm{x}^2 + 16\mathrm{xy} - 4\mathrm{y}^2,$ $6\mathrm{x} - 3\mathrm{y} + 4\mathrm{x}^2 - 4\mathrm{xy} + \mathrm{y}^2,$ $-6\mathrm{x} + 3\mathrm{y} - 4\mathrm{x}^2 + 4\mathrm{xy} - \mathrm{y}^2,$ $36\mathrm{x} - 64\mathrm{x}^2 - 80\mathrm{xy} + 20\mathrm{y}^2 + 216\mathrm{xcount} - 81\mathrm{count}^2,$ $-36\mathrm{x} + 64\mathrm{x}^2 + 80\mathrm{xy} - 20\mathrm{y}^2 - 216\mathrm{xcount} + 81\mathrm{count}^2,$ $9 - 4\mathrm{x}^2 + 4\mathrm{xy} - \mathrm{y}^2,$ $-9 + 4\mathrm{x}^2 - 4\mathrm{xy} + \mathrm{y}^2,$ $8\mathrm{x}^2 + 4\mathrm{xy} - 4\mathrm{y}^2 - 18\mathrm{xcount} + 9\mathrm{ycount},$ $-8\mathrm{x}^2 - 4\mathrm{xy} + 4\mathrm{y}^2 + 18\mathrm{xcount} - 9\mathrm{ycount}$	$c_0 - c_1 + c_2 + c_4 - c_5 + c_6 \geq 0,$ $12c_1 + 24c_2 + 16c_3 + 3c_4 + 6c_5 + 12c_6 \geq 0,$ $12c_1 + 24c_2 + 16c_3 + 270c_4 + 273c_5 + 12c_6 + 178c_7 = 0,$ $24c_1 + 48c_2 + 32c_3 + 6c_4 + 279c_5 + 1092c_6 + 356c_8 = 0,$ $27c_1 + 54c_2 + 36c_3 + 207c_4 + 414c_5 + 828c_6 - 356c_9 = 0$

The interpretation of these results is as follows: for any generator expression `e` inside the fixed point, $(\forall n \geq 0)\ \mathbb{E}(\mathtt{e} \mid n) \geq 0$.

Finally, MOT-EX-POLY represents the original motivating program of Example 1.1 where we force our analysis to track all monomials over the program variables upto degree 2. This is done by introducing a fresh variable for each monomial. Using this approach we were able to show interesting *polynomial* inductive expectation invariants. For example, when we combine generator expressions 5 and 6, we obtain the following fact: $\mathbb{E}((2x - y)^2 \mid n) = \mathbb{E}(3(2x - y) \mid n)$ for all $n \geq 0$. This fact could be use to reduce the complexity of a quadratic expression to a linear one in a larger static analysis framework.

6.3 Other Benchmark Examples

6.3.1 2d-walk

Example 6.2. The program in Figure 4 simulates a peg performing an infinite random walk in 2 dimensional space. The peg starts at the origin $(0, 0)$ and first picks uniformly at random a direction $\{1 : E; 2 : N; 3 : W; 4 : S\}$ and a uniform random displacement $r_1 \sim \mathtt{unifRand}(0, 2)$. Depending on which quadrant the peg is in, the values of the (x, y) cooradinates and distance from origin are updated accurately.

Our analysis was able to show the following IEI: $\mathbb{E}(\mathrm{count} \mid n) \geq 0$, $\mathbb{E}(\mathrm{x} \mid n) = 0$, $\mathbb{E}(\mathrm{y} \mid n) = 0$, and $\mathbb{E}(\mathrm{dist} \mid n) = 0$, for all $n \geq 0$. This confirms the intuition and theory on random walks that in expectation the peg does not drift, in neither direction nor total distance, far away the origin.

6.3.2 aggregate-rv

Example 6.3. The program in Figure 5 simulates the aggregation of 500 independent identically distributed draws from a uniform random distribution over the interval [0,1]. Our analysis was able to generate the following inductive expectation invariants: $\mathbb{E}(\mathrm{N} \mid n) = 500$, $\mathbb{E}(\mathrm{i} - 2\mathrm{x} \mid n) = 0$, for all $n \geq 0$. Combining the IEI i - 2x with the additional loop invariant $\mathrm{i} \leq \mathrm{N}$ we arrive at the $\mathbb{E}(\mathrm{x}) = 250$ at the end of the loop.

6.3.3 hare-turtle

Example 6.4. The program in Figure 6 models a stochastic version of Aesop's fable about the turtle and the hare. In this setting the hare chooses to wait until the turtle gains some significant advantage ($t = 30$ initially) before starting to ponder whether to dash. At each time step the turtle proceeds at unit pace, whereas the hare first chooses uniformly at random whether to dash at this time instance. If he chooses to dash, his paces is a uniform draw over $[0, 10]$.

```
real x,y, dist, count
while (true) {
   c = choice (1:1/4, 2:1/4, 3:1/4, 4:1/4)
   if (x >= 0){
     if (y >= 0) {
       switch (c){
          1: x,dist := (x + r1, dist + r1)
          2: y,dist := (y + r1, dist + r1)
          3: x,dist := (x - r1, dist - r1)
          4: y,dist := (y - r1, dist - r1)
        }
     } else{
         switch (c){
          1: x,dist := (x + r1, dist + r1)
          2: y,dist := (y + r1, dist - r1)
          3: x,dist := (x - r1, dist - r1)
          4: y,dist := (y - r1, dist + r1)
        }
     }
   } else {
     if (y >= 0) {
       switch (c){
          1: x,dist := (x - r1, dist + r1)
          2: y,dist := (y + r1, dist + r1)
          3: x,dist := (x + r1, dist - r1)
          4: y,dist := (y - r1, dist - r1)
        }
     } else{
         switch (c){
          1: x,dist := (x - r1, dist + r1)
          2: y,dist := (y + r1, dist - r1)
          3: x,dist := (x + r1, dist - r1)
          4: y,dist := (y - r1, dist + r1)
        }
     }

   }
   count := count + 1;
}
```

$$
\begin{array}{lll}
\mathbf{g}_1 & : & (x \geq 0) \wedge (y \geq 0) \\
\mathcal{F}_{\tau_1} & : & \left\{ \begin{array}{lll}
f_{11}: & \left[\begin{array}{lll} \text{x'} & \mapsto & \text{x} + r_1, \\ \text{dist'} & \mapsto & \text{dist} + r_1, \\ \text{count'} & \mapsto & \text{count} + 1, \end{array}\right] & \text{w.p. } \frac{1}{4} \\
f_{12}: & \left[\begin{array}{lll} \text{y'} & \mapsto & \text{y} + r_1, \\ \text{dist'} & \mapsto & \text{dist} + r_1, \\ \text{count'} & \mapsto & \text{count} + 1, \end{array}\right] & \text{w.p. } \frac{1}{4} \\
f_{13}: & \left[\begin{array}{lll} \text{x'} & \mapsto & \text{x} - r_1, \\ \text{dist'} & \mapsto & \text{dist} - r_1, \\ \text{count'} & \mapsto & \text{count} + 1, \end{array}\right] & \text{w.p. } \frac{1}{4} \\
f_{14}: & \left[\begin{array}{lll} \text{y'} & \mapsto & \text{y} - r_1, \\ \text{dist'} & \mapsto & \text{dist} - r_1, \\ \text{count'} & \mapsto & \text{count} + 1, \end{array}\right] & \text{w.p. } \frac{1}{4}
\end{array} \right. \\
\mathbf{g}_2 & : & (x \geq 0) \wedge (y \leq 0) \\
\mathcal{F}_{\tau_2} & : & \left\{ \begin{array}{lll}
f_{21}: & \left[\begin{array}{lll} \text{x'} & \mapsto & \text{x} + r_1, \\ \text{dist'} & \mapsto & \text{dist} + r_1, \\ \text{count'} & \mapsto & \text{count} + 1, \end{array}\right] & \text{w.p. } \frac{1}{4} \\
f_{22}: & \left[\begin{array}{lll} \text{y'} & \mapsto & \text{y} + r_1, \\ \text{dist'} & \mapsto & \text{dist} - r_1, \\ \text{count'} & \mapsto & \text{count} + 1, \end{array}\right] & \text{w.p. } \frac{1}{4} \\
f_{23}: & \left[\begin{array}{lll} \text{x'} & \mapsto & \text{x} - r_1, \\ \text{dist'} & \mapsto & \text{dist} - r_1, \\ \text{count'} & \mapsto & \text{count} + 1, \end{array}\right] & \text{w.p. } \frac{1}{4} \\
f_{24}: & \left[\begin{array}{lll} \text{y'} & \mapsto & \text{y} - r_1, \\ \text{dist'} & \mapsto & \text{dist} + r_1, \\ \text{count'} & \mapsto & \text{count} + 1, \end{array}\right] & \text{w.p. } \frac{1}{4}
\end{array} \right. \\
\mathbf{g}_3 & : & (x \leq 0) \wedge (y \geq 0) \\
\mathcal{F}_{\tau_3} & : & \left\{ \begin{array}{lll}
f_{31}: & \left[\begin{array}{lll} \text{x'} & \mapsto & \text{x} + r_1, \\ \text{dist'} & \mapsto & \text{dist} - r_1, \\ \text{count'} & \mapsto & \text{count} + 1, \end{array}\right] & \text{w.p. } \frac{1}{4} \\
f_{32}: & \left[\begin{array}{lll} \text{y'} & \mapsto & \text{y} + r_1, \\ \text{dist'} & \mapsto & \text{dist} + r_1, \\ \text{count'} & \mapsto & \text{count} + 1, \end{array}\right] & \text{w.p. } \frac{1}{4} \\
f_{33}: & \left[\begin{array}{lll} \text{x'} & \mapsto & \text{x} - r_1, \\ \text{dist'} & \mapsto & \text{dist} + r_1, \\ \text{count'} & \mapsto & \text{count} + 1, \end{array}\right] & \text{w.p. } \frac{1}{4} \\
f_{34}: & \left[\begin{array}{lll} \text{y'} & \mapsto & \text{y} - r_1, \\ \text{dist'} & \mapsto & \text{dist} - r_1, \\ \text{count'} & \mapsto & \text{count} + 1, \end{array}\right] & \text{w.p. } \frac{1}{4}
\end{array} \right. \\
\mathbf{g}_4 & : & (x \leq 0) \wedge (y \leq 0) \\
\mathcal{F}_{\tau_4} & : & \left\{ \begin{array}{lll}
f_{41}: & \left[\begin{array}{lll} \text{x'} & \mapsto & \text{x} + r_1, \\ \text{dist'} & \mapsto & \text{dist} - r_1, \\ \text{count'} & \mapsto & \text{count} + 1, \end{array}\right] & \text{w.p. } \frac{1}{4} \\
f_{42}: & \left[\begin{array}{lll} \text{y'} & \mapsto & \text{y} + r_1, \\ \text{dist'} & \mapsto & \text{dist} - r_1, \\ \text{count'} & \mapsto & \text{count} + 1, \end{array}\right] & \text{w.p. } \frac{1}{4} \\
f_{43}: & \left[\begin{array}{lll} \text{x'} & \mapsto & \text{x} - r_1, \\ \text{dist'} & \mapsto & \text{dist} + r_1, \\ \text{count'} & \mapsto & \text{count} + 1, \end{array}\right] & \text{w.p. } \frac{1}{4} \\
f_{44}: & \left[\begin{array}{lll} \text{y'} & \mapsto & \text{y} - r_1, \\ \text{dist'} & \mapsto & \text{dist} + r_1, \\ \text{count'} & \mapsto & \text{count} + 1, \end{array}\right] & \text{w.p. } \frac{1}{4}
\end{array} \right.
\end{array}
$$

Figure 4: 2D-RANDOM-WALK (**Left**) Program modeling a peg on a random walk in 2D. (**Right**) The corresponding PWL probabilistic loop.

Given this setup our analysis was able to show the following IEI: $\mathbb{E}(5\text{t - }2\text{h} \mid n) \geq 0$, $\mathbb{E}(2\text{h - }5\text{t + }50 \mid n) = 0$, for all $n \geq 0$. This translates that in expectation (over the hare's choice to dash or not and hare's pace at each instant), the turtle is no more than 2.5 times slower than the hare. Additionally, 2h - 5t + 50 is a quantity that is conserved in expectation across any number $n \geq 0$ of iterations of the loop.

6.3.4 coupon5

Example 6.5. The program in Figure 7 illustrates the Coupon Collector's Problem: Given $n = 5$ distinct coupons, by choosing (with replacement) uniformly at random a coupon, what is the expected number of coupons (count) you should draw before collecting all $n = 5$ coupons ($i = n$). This is a well known problem where the expected number of coupons to draw grows as $\Theta(n \log n)$ in the number of coupons n.

For a linearized version of the problem, our analysis showed the following inductive expectation invariants: $\mathbb{E}(1.111\text{i} - \text{count} \mid n) \geq 0$, $\mathbb{E}(\text{count} - \text{i} \mid n) \geq 0$, $\mathbb{E}(4\text{count} - 5\text{i} + 1 \mid n) = 0$, for all $n \geq 0$.

```
real x = 0;
real N = 500;
for ( i=0; i < N; ++i )
    x = x + unifRand(0,1)
```

$$
\begin{array}{lcl}
\mathbf{g}_1 & : & (i \leq N) \\
\mathcal{F}_{\tau_1} & : & \left\{ f_{11} : \left[\begin{array}{lcl} \text{x'} & \mapsto & \text{x} + r_1, \\ \text{i'} & \mapsto & \text{i} + 1, \\ \text{N'} & \mapsto & \text{N}, \end{array} \right] \right. \quad \text{w.p. } 1 \\
\mathbf{g}_2 & : & (i > N) \\
\mathcal{F}_{\tau_2} & : & \left\{ f_{21} : \left[\begin{array}{lcl} \text{x'} & \mapsto & \text{x}, \\ \text{i'} & \mapsto & \text{i}, \\ \text{N'} & \mapsto & \text{N}, \end{array} \right] \right. \quad \text{w.p. } 1
\end{array}
$$

Figure 5: AGGREGATE-RV (**Left**) Probabilistic program that accumulates 500 draws from a uniform random distribution. (**Right**) The corresponding PWL probabilistic loop.

```
real h, t;
// h is hare and t is tortoise h=0;t=30;
while(h<=t){
    if (flip (0.5) )
        h = h + unifRand(0,10);
    t = t + 1;
```

$$
\begin{array}{lcl}
\mathbf{g}_1 & : & (h \leq t) \\
\mathcal{F}_{\tau_1} & : & \left\{ \begin{array}{l} f_{11} : \left[\begin{array}{lcl} \text{h'} & \mapsto & \text{h} + r_1, \\ \text{t'} & \mapsto & \text{t} + 1, \end{array} \right] \quad \text{w.p. } \frac{1}{2} \\ f_{12} : \left[\begin{array}{lcl} \text{h'} & \mapsto & \text{h}, \\ \text{t'} & \mapsto & \text{t} + 1, \end{array} \right] \quad \text{w.p. } \frac{1}{2} \end{array} \right. \\
\mathbf{g}_2 & : & (h > t) \\
\mathcal{F}_{\tau_2} & : & \left\{ f_{21} : \left[\begin{array}{lcl} \text{h'} & \mapsto & \text{h}, \\ \text{t'} & \mapsto & \text{t}, \end{array} \right] \right. \quad \text{w.p. } 1
\end{array}
$$

Figure 6: HARE-TURTLE (**Left**) Probabilistic program that models a stochastic version of the hare vs. turtle fable. h is whimsical in deciding when to move but quick; t is slow but consistent. (**Right**) The corresponding PWL probabilistic loop.

6.3.5 hawk-dove

Example 6.6. Figure 8 shows a model of a simple 2-player *stochastic evolutionary game*. In this game each player chooses one of two possible strategies: a collaborative strategy Dove or an aggressor strategy Hawk. If both players collaborate, they are guaranteed to split resources equally. In this setting Hawk is a strictly dominant strategy, which guarantees full amount of resource for the player in case opponent plays Dove. If both players attempt to use Hawk strategy then the outcome is stochastic: one player gets all resources while the other suffers penalty (injury).

At every round of the game the players carefully weight the penalty for losing a fight c against the common resources v for the round and perform a stochastic choice for a strategy (player$_i$). Next, both strategies are evaluated and the corresponding balance of resources (p$i$bal) is updated.

In the case of a fair balance of penalty to common resource ration (in HAWK-DOVE-FAIR), our analysis generates the following inductive expectation invariants: $\mathbb{E}(\text{count} \mid n) \geq 0$, $\mathbb{E}(\text{count} - \text{p1bal} \mid n) = 0$, $\mathbb{E}(\text{count} - \text{p2bal} \mid n) = 0$, $\mathbb{E}(\text{p2gain} \mid n) = 0$, for all $n \geq 0$. The last IEI shows we can detect that the probabilistic choice of strategy as stated is indeed an evolutionary (stochastically) stable strategy that is also fair (no gain over other player). This last IEI cannot be established when the balance of c to v is broken (demonstrated in HAWK-DOVE-BIAS).

```
int count,i
i:= 1
count := 0
while (i>=1 & i<=2)
    count := count + 1
    if (flip(4/5))
        i := i + 1
    break
//Collected second coupon
while (i>=2 & i<=3)
    count := count + 1
    if (flip(3/5))
        i := i + 1
    break
//Collected third coupon

while (i>=3 & i<=4)
    count := count + 1
    if (flip(2/5))
        i := i + 1
//Collected fourth coupon

while (i>=4 & i<=5)
    count := count + 1
    if (flip(1/5))
        i := i + 1
//Collected all coupons
```

$$
\begin{array}{lcl}
\mathbf{g}_1 & : & (1 \le i) \wedge (i \le 2) \\
\mathcal{F}_{\tau_1} & : & \begin{cases} f_{11} : \begin{bmatrix} \text{i'} & \mapsto & \text{i} + 1, \\ \text{count'} & \mapsto & \text{count} + 1, \end{bmatrix} & \text{w.p. } \frac{4}{5} \\ f_{12} : \begin{bmatrix} \text{i'} & \mapsto & \text{i}, \\ \text{count'} & \mapsto & \text{count} + 1, \end{bmatrix} & \text{w.p. } \frac{1}{5} \end{cases} \\
\mathbf{g}_2 & : & (2 \le i) \wedge (i \le 3) \\
\mathcal{F}_{\tau_2} & : & \begin{cases} f_{21} : \begin{bmatrix} \text{i'} & \mapsto & \text{i} + 1, \\ \text{count'} & \mapsto & \text{count} + 1, \end{bmatrix} & \text{w.p. } \frac{3}{5} \\ f_{22} : \begin{bmatrix} \text{i'} & \mapsto & \text{i}, \\ \text{count'} & \mapsto & \text{count} + 1, \end{bmatrix} & \text{w.p. } \frac{2}{5} \end{cases} \\
\mathbf{g}_3 & : & (3 \le i) \wedge (i \le 4) \\
\mathcal{F}_{\tau_3} & : & \begin{cases} f_{31} : \begin{bmatrix} \text{i'} & \mapsto & \text{i} + 1, \\ \text{count'} & \mapsto & \text{count} + 1, \end{bmatrix} & \text{w.p. } \frac{2}{5} \\ f_{32} : \begin{bmatrix} \text{i'} & \mapsto & \text{i}, \\ \text{count'} & \mapsto & \text{count} + 1, \end{bmatrix} & \text{w.p. } \frac{3}{5} \end{cases} \\
\mathbf{g}_4 & : & (4 \le i) \wedge (i \le 5) \\
\mathcal{F}_{\tau_4} & : & \begin{cases} f_{41} : \begin{bmatrix} \text{i'} & \mapsto & \text{i} + 1, \\ \text{count'} & \mapsto & \text{count} + 1, \end{bmatrix} & \text{w.p. } \frac{1}{5} \\ f_{42} : \begin{bmatrix} \text{i'} & \mapsto & \text{i}, \\ \text{count'} & \mapsto & \text{count} + 1, \end{bmatrix} & \text{w.p. } \frac{4}{5} \end{cases} \\
\mathbf{g}_5 & : & (5 < i) \\
\mathcal{F}_{\tau_5} & : & \begin{cases} f_{51} : \begin{bmatrix} \text{i'} & \mapsto & \text{i}, \\ \text{count'} & \mapsto & \text{count}, \end{bmatrix} & \text{w.p. } 1 \end{cases}
\end{array}
$$

Figure 7: COUPON5 (**Left**) Probabilistic program that models the Coupon Collector Problem for $n = 5$ coupons. (**Right**) The corresponding PWL probabilistic loop.

```
real p1bal,p2bal = 0, 0
count = 0
v = 4 // common resource, objective
c = 8 // penalty for losing a fight
while (true) { // one could choose a finite #rounds
   player1 = choice (1:(c-v)/2, 2:v/2)
   player2 = choice (1:(c-v)/2, 2:v/2)
   if (player1 <= 1) //player1 Dove
     if (player2 <= 1)  //player2 Dove
       //collaborators split resources
       p1bal := p1bal + v/2
       p2bal := p2bal + v/2
     else //player2 Hawk
       //hawk strategy > dove, winner takes all
       p2bal := p2bal + v
       //dove avoids fight
       p1bal := p1bal
   else //player1 Hawk
     if (player2 <= 1) //player2 Dove
       //hawk strategy > dove, winner takes all
       p1bal := p1bal + v
       //dove avoids fight
       p2bal := p2bal
     else //player2 Hawk
       //both players fight, winner takes all
       //loser suffers penalty
       //outcome of fight is stochastic
       if flip(1/2)
            p1bal := p1bal + v
            p2bal := p2bal - c
        else
            p1bal := p1bal - c
            p2bal := p2bal + v
   count := count + 1;
}
```

$$
\begin{array}{rcl}
\mathbf{g}_1 & : & (0 \geq 1) \\
\mathcal{F}_{\tau_1} & : & \left\{ \begin{array}{lll}
f_{11} : \begin{bmatrix} \text{count'} & \mapsto & \text{count} + 1, \\ \text{p1bal'} & \mapsto & \text{p1bal} + \frac{\text{v}}{2}, \\ \text{p2bal'} & \mapsto & \text{p2bal} + \frac{\text{v}}{2}, \\ \text{p2gain'} & \mapsto & \text{p2gain}, \end{bmatrix} & \text{w.p. } \frac{16}{64} \\
f_{12} : \begin{bmatrix} \text{count'} & \mapsto & \text{count} + 1, \\ \text{p1bal'} & \mapsto & \text{p1bal}, \\ \text{p2bal'} & \mapsto & \text{p2bal} + \text{v}, \\ \text{p2gain'} & \mapsto & \text{p2gain} + \text{v}, \end{bmatrix} & \text{w.p. } \frac{16}{64} \\
f_{13} : \begin{bmatrix} \text{count'} & \mapsto & \text{count} + 1, \\ \text{p1bal'} & \mapsto & \text{p1bal} + \text{v}, \\ \text{p2bal'} & \mapsto & \text{p2bal}, \\ \text{p2gain'} & \mapsto & \text{p2gain} - \text{v}, \end{bmatrix} & \text{w.p. } \frac{16}{64} \\
f_{14} : \begin{bmatrix} \text{count'} & \mapsto & \text{count} + 1, \\ \text{p1bal'} & \mapsto & \text{p1bal} + \text{v}, \\ \text{p2bal'} & \mapsto & \text{p2bal} - \text{c}, \\ \text{p2gain'} & \mapsto & \text{p2gain} - \text{c} - \text{v}, \end{bmatrix} & \text{w.p. } \frac{8}{64} \\
f_{15} : \begin{bmatrix} \text{count'} & \mapsto & \text{count} + 1, \\ \text{p1bal'} & \mapsto & \text{p1bal} - \text{c}, \\ \text{p2bal'} & \mapsto & \text{p2bal} + \text{v}, \\ \text{p2gain'} & \mapsto & \text{p2gain} + \text{c} + \text{v}, \end{bmatrix} & \text{w.p. } \frac{8}{64}
\end{array} \right.
\end{array}
$$

Figure 8: HAWK-DOVE (**Left**) Probabilistic program that models a stochastic 2-player evolutionary game of Dove and Hawk. (**Right**) The corresponding PWL probabilistic loop.