Expectation Invariants for Probabilistic Program Loops as Fixed Points

Seminar on Probabilistic Programs

Phillip Florian (308704)

Informatik 2 RWTH Aachen

February 5, 2015

1 Introduction

Probabilistic programs have a large field of applications. They are used in security [4], randomized algorithms [5], machine learning [?], for verification of matrix multiplications [5] and basically in any application area that relies on probabilistic choices. A probabilistic program is a program that may toss a (possibly unfair) coin or draw variables from random distributions such that the further execution of the program is influenced by these probabilistic choices [3]. This randomness of the programs behaviour makes it hard to verify properties of such programs. This is where expectation invariants come into play. They can be used as an abstract interpretation for analyzing probabilistic programs e.g., to prove the correctness of such a program.

An example probabilistic program is shown in Figure 1. The function $flip(3/4)$ denotes a coin-flip where one side has a probability of $\frac{3}{4}$ and the other one has a probability of $\frac{1}{4}$. The function $rand(-5, 3)$ is the function that draws an integer from the interval $[-5, 3]$, where all values have the same probability of being drawn. It is easy to see that there are many different ways an execution of the program may look like. Running the program multiple times and analyzing the value "count", which counts the number of loop iterations until termination, it can be shown that the loop takes on average 5.1345 iterations until it terminates [1]. In practice it is not always easy to execute a program loop a lot of times, i.e., because of a very high run-time, to get an approximation on the expected values. That is exactly where expectation invariants come into play.

```
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++
}
```

Figure 1: Example of a probabilistic program

This work is based on [1, 2] and it will explain how fixed points for expectation invariants can be computed by an iteration over cones of expressions. We will first briefly explain what probabilistic programs are and how a probabilistic loop in such a program is defined. Afterwards, the concept of pre-expectations will be introduced for computing the expected values in the next iteration. Finally we will show how to obtain expectation invariants fixed points of a probabilistic loop by iterative algorithm.

2 Preliminaries

2.1 Probabilistic Programs

Any program that draws variables from random distributions, e.g. a uniform distribution or a Gaussian distribution, is a probabilistic program [3]. In the following let $\mathcal{P}$ be a probabilistic program written in an imperative language using random number generators like choosing an integer from a uniform distribution. Further, let $X = \{x_1, \ldots, x_m\}$ be a finite set of real-valued program variables and let $R = \{r_1, \ldots, r_l\}$ be a finite set of real-valued random variables occurring in $\mathcal{P}$. In addition, let $\boldsymbol{x}$ and $\boldsymbol{r}$ be vectors denoting the valuations of all program and random variables respectively. The random variables in R have a joint distribution $\mathcal{D}_R$. Formally, the distribution is defined over an underlying σ-algebra with an appropriate measure μ_R.

In this seminar work, we are especially interested in analyzing the loops occurring in a probabilistic program. Thus we only need to know the program values at the beginning of the loop and the changes to these variables occurring inside the loop.

Definition 2.1 (Probabilistic loops)**.** A probabilistic loop of $\mathcal{P}$ is a tuple $\langle \mathcal{T}, \mathcal{D}_0, n\rangle$, wherein $\mathcal{T} : \{\tau_1, \ldots, \tau_k\}$ is a finite set of probabilistic transitions (from the loop head to itself). $\mathcal{D}_0$ is the initial probability distribution of the program variables and n is a loop counter variable

```
real x := rand (−5,3)
while (true){
  if(x<=10)
    x:= x + rand (0,2)
  else
    do nothing
}
```

Figure 2: Simple example for a probabilistic loop

introduced to keep track of the number of loop iterations.
Each probabilistic transition $\tau_i : \langle \mathbf{g}_i, \mathcal{F}_i \rangle$ consists of a guard $\mathbf{g}_i[X]$ over X, i.e. a Boolean condition on X, and an update function $\mathcal{F}_i(\boldsymbol{x}, \boldsymbol{r})$ such that for the program values $\boldsymbol{x}'$ after taking the transition it holds: $\boldsymbol{x}' = \mathcal{F}_i(\boldsymbol{x}, \boldsymbol{r})$.

Note that the loop counter is not part of the program variables, but there can be variables inside the loop counting the number of iterations. Also note that in every iteration $\boldsymbol{r}$ is drawn from the joint distribution $\mathcal{D}_R$ of random variables.

Example: Consider the program code shown in Figure 2. The loop of the program can be written as a probabilistic loop with a single transition $\tau : \langle \mathbf{g}, \mathcal{F} \rangle$, with $\mathbf{g} = x \leq 10$ and $\mathcal{F}(\boldsymbol{x}, \boldsymbol{r}) = x + r$, where $r = U(0, 2)$ is drawn from the uniform distribution over the interval $[0, 2]$. Further, the initial program variables $\boldsymbol{x}$ are drawn from $D_0 : \langle x \rangle = U[-5, 3]$ and initially $n = 0$.

■

All methods explained in this seminar work are restricted to piecewise linear probabilistic loops, where each transition τ_i has a linear guard and piecewise linear transitions (defined below). Furthermore, this seminar only covers expectation invariants over simple loops (no nesting). For an explanation of how those methods can be used to compute expectation invariants for nested loops we refer to [2].

A piecewise linear probabilistic loop is defined like a probabilistic loop except for the probabilistic transitions which are now piecewise linear.

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

- $\mathbf{g}$ is a linear guard over X

- $\mathcal{F}(\boldsymbol{x}, \boldsymbol{r})$ is a piecewise linear function for X, i.e. all parts of the function are linear, where $\boldsymbol{r}$ is decomposed into a vector of continuous random choices $\boldsymbol{r}_c$ and a vector of discrete Bernoulli choices $\boldsymbol{r}_b$. As a result the update function $\mathcal{F}(\boldsymbol{x}, \boldsymbol{r})$ may be written as:

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

where $f_1, \ldots, f_k$ denote the different outcomes of the Bernoulli choices in $\boldsymbol{r}_b$ and are used as indicators of the different forks. The values $p_1, \ldots, p_k$ represent the probabilities with which the corresponding fork is chosen, where $0 < p_i \leq 1$ and $\sum_{i=1}^{k} p_i = 1$. Further, the matrices $A_1, \ldots, A_k \in \mathbb{R}^{m \times m}$, $B_1, \ldots, B_k \in \mathbb{R}^{m \times l}$ and vectors $d_1, \ldots, d_k \in \mathbb{R}^m$ are used to model the changes to the program variables occurring in the loop.

Example: The piecewise linear transition $\tau : \langle \mathbf{g}, \mathcal{F} \rangle$ corresponding to the loop from Figure 3 (left) can be expressed as:

- $\mathbf{g} : (x + y \leq 10)$
- $\mathcal{F}(\boldsymbol{x}, \boldsymbol{r}) : \begin{cases} f_1 : \begin{pmatrix} x \\ y \\ count \end{pmatrix} + \begin{pmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} r_1 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 2 \\ 1 \end{pmatrix} & , p_1 = \frac{3}{4} \\ f_2 : \begin{pmatrix} x \\ y \\ count \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} & , p_2 = \frac{1}{4} \end{cases}$

where $r_1 = U[0, 2]$.

■

For a Piecewise linear loop we preclude non-determinism of a probabilistic loop $\langle \mathcal{T}, \mathcal{D}_0, n \rangle$ by imposing two restrictions for all transitions $\tau \in \mathcal{T}$:

1. Mutual exclusion: for all $\tau_i : \langle \mathbf{g}_i, \mathcal{F}_i \rangle$ and $\tau_j : \langle \mathbf{g}_j, \mathcal{F}_j \rangle$ in $\mathcal{T}$ with $\tau_i \neq \tau_j$ it holds $\mathbf{g}_i \wedge \mathbf{g}_j \equiv \text{false}$
2. Exhaustiveness: $\bigvee_{\tau_i \in \mathcal{T}} \mathbf{g}_i \equiv \text{true}$

These two restrictions guarantee that precisely one transition can be taken at any iteration of the loop.

To guarantee that exactly one transition can be taken at every iteration we might have to change the loop before so that it becomes infinitely looping. If the loop is not already able

```
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 (true){
  if(x+y<=10)
    if flip(3/4)
      x:= x + rand (0,2)
      y:= y + 2
    count++
  else
    do nothing
}
```

Figure 3: Example of a probabilistic program (left) and its stuttering version (right)

to perform infinitely many iteration, it has to modified such that all program variables are preserved in each iteration once the original loop condition is violated. We call this new loop the stuttering version of the original loop. It is obtained by creating an infinite loop with an if-statement inside. If the original loop-guard is satisfied, then the body of the original loop is executed. Else, nothing happens.

Example: Consider the code from Figure 3 (left). The corresponding stuttering code Figure 3 (right) is obtained by transforming the loop to an endless loop with an if-statement inside the loop. The first part is for the original loop that is taken if the original loop-guard is valid. It executes the original loop body. The second part is used to preserve the program variables once the loop-invariant is violated.

■

Example: (Taken from [2]) Take a look at the stuttering version of the program discussed before (right side of Figure 3) where the initial values of the program variables reaching the loop head are drawn from the initial distribution $\mathcal{D}_0 : U[-5,3] \times U[-3,5] \times \{0\}$. The stuttering adds a new path inside the loop that preserves the values of the program variables once the loop guard $x + y \leq 10$ is violated. The probabilistic program now has two transitions $\tau_1 = \langle \mathbf{g}_1, \mathcal{F}_1 \rangle, \tau_2 = \langle \mathbf{g}_2, \mathcal{F}_2 \rangle$, where τ_1 represents the loop body of the original code and τ_2 preserves the program variables. Looking at the code we get:

$$\begin{array}{l|l}
\mathbf{g}_1 : (x + y \leq 10) & \mathbf{g}_2 : (x + y > 10) \\
\mathcal{F}_1 : \begin{cases} f_1 : \begin{bmatrix} x' & \mapsto x + r_1 \\ y' & \mapsto y + 2 \\ count' & \mapsto count + 1 \end{bmatrix} & w.p. \frac{3}{4} \\ f_2 : \begin{bmatrix} x' & \mapsto x \\ y' & \mapsto y \\ count' & \mapsto count + 1 \end{bmatrix} & w.p. \frac{1}{4} \end{cases} & \mathcal{F}_2 : \begin{cases} x' & \mapsto x \\ y' & \mapsto y \\ count' & \mapsto count \end{cases}
\end{array}$$

where r_1 represents the uniform random variable over $[0, 2]$.

■

To model an execution of a probabilistic loop we use tuples $(\boldsymbol{x}_n, n)$ as states of the loop, where $\boldsymbol{x}_n$ denotes the vector representing the program variables at loop-iteration n. Further, let $(\boldsymbol{x}_0, 0)$ be an initial state if $\boldsymbol{x}_0$ is a sample drawn from the initial distribution $\mathcal{D}_0$. $(\boldsymbol{x}_i, i)$ is a predecessor of $(\boldsymbol{x}_{i+1}, i+1)$ if for a transition τ, $\boldsymbol{x}_i \models \mathbf{g}$ and there exists a $\boldsymbol{r} \in \mathcal{D}_R$ s.t. $\boldsymbol{x}_{i+1} = \mathcal{F}(\boldsymbol{x}_i, \boldsymbol{r})$. $\mathcal{D}_i$ denotes the distribution of program values after i iterations and can be seen as the set of possible states after i iterations.

Definition 2.3 (Sample path)**.** A sample path (also referred to as an execution) of a loop is an infinite sequence of states $(\boldsymbol{x}_0, 0) \xrightarrow{\tau_0, \boldsymbol{r}_0} (\boldsymbol{x}_1, 1) \xrightarrow{\tau_1, \boldsymbol{r}_1} (\boldsymbol{x}_2, 2) \ldots$, where $(X_0$ is a sample from $\mathcal{D}_0$ and for each $i \geq 0$, $(\boldsymbol{x}_{i+1}, i+1)$ is obtained by executing the unique transition $\tau_i : (\mathbf{g}_i.\mathcal{F}_i)$ that is enabled in the state $(\boldsymbol{x}_i, i)$. This execution involves a sample from the Bernoulli random variables to choose a fork of the transition τ_i and a choice of the continuous random variables $\boldsymbol{r}_c$ to obtain $\boldsymbol{x}_{i+1} = \mathcal{F}_i(\boldsymbol{x}_i, \boldsymbol{r}_i)$.

Example: One possible execution of the PWL loop discussed in the previous example would be

$$((3,3,0)^T, 0) \xrightarrow{\tau_1, r_1 = 1} ((4,5,1)^T, 1) \xrightarrow{\tau_1, r_1 = 2}$$
$$((6,7,2)^T, 2) \xrightarrow{\tau_2} ((6,7,2)^T, 3) \xrightarrow{\tau_2} \ldots$$

■

2.2 Pre-Expectations

In this section we define the concept of pre-expectations for an affine expression e over program variables across a fixed transition τ. We will then refine this definition for PWL transitions and use this to define the pre-expectation operator with respect to all transitions of a probabilistic loop. In the next section this will be lifted to pre-expectations of cones over affine expressions and later on be used to define the operator used for finding expectation invariants as fixed points of this operator.

We are going to reason only about affine expressions e, i.e. linear expressions plus a constant term, where $e = c_0 + \sum_{i=0}^{m} \lambda_i \cdot x_i$, $c_0, \lambda_i \in \mathbb{R}$. Further, $e[\boldsymbol{x}_i]$ denotes the expression involving the program variables at step i.
Given any state $(\boldsymbol{x}, i)$ of the loop and an affine expression $e[\boldsymbol{x}]$,involving the program variables, we want to compute the expected value of the expression $e[\boldsymbol{x}']$ evaluated over all possible successor states $(\boldsymbol{x}', n+1)$ that can be reached in one loop iteration starting from $(\boldsymbol{x}, n)$. The function that maps $\boldsymbol{x}$ to the expectation of the expression e in the next step is called the pre-expectation of e.

Definition 2.4 (Pre-expectation for fixed transitions)**.** Let $\tau : \langle \mathbf{g}, \mathcal{F} \rangle$ be a transition and $e[\boldsymbol{x}]$ be an affine expression involving the set of program variables $\boldsymbol{x}$ of a probabilistic program $\mathcal{P}$. The pre-expectation operator $\text{pre}\mathbb{E}_\tau$ is an expression transformer that associates each expression $e[\boldsymbol{x}]$ with the expectation of the expression $e[\boldsymbol{x}']$ obtained by taking transition τ. Formally

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

where $\mathbb{E}_R$ is taken over the distribution $\mathcal{D}_R$.

If we now consider a PWL transition τ with forks $f_1, \dots, f_k$ and with fork probabilities $p_1, \dots, p_k$ $k > 0$, we are able to simplify the computation of $\text{pre}\mathbb{E}_\tau$.

Definition 2.5 (Pre-expectation for fixed PWL transitions)**.** For a PWL transition τ the pre-expectation operator can be written as

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

where $\text{pre}(e', f_j)$ denotes the expression obtained by substituting all variables of $\boldsymbol{x}'$ occurring in e' with their corresponding values obtained by applying f_j. The expectation $\mathbb{E}_{\mathcal{R}}(d)$ denotes the expectation of d over the joint distribution $\mathcal{D}_R$ of the random variables.

Example: We again consider the program from Figure 3 (right) and will demonstrate the notion of a pre-expectation by considering the expression $e : 1 + 2x - 3y$ across transition $\tau_1 : \langle \mathbf{g}_1, \mathcal{F}_1 \rangle$ with $\mathbf{g}_1 : x + y \leq 10$ and

$$\mathcal{F}_1 : \begin{cases} f_1 : \begin{bmatrix} x' & \mapsto x + r_1 \\ y' & \mapsto y + 2 \\ \mathit{count}' & \mapsto \mathit{count} + 1 \end{bmatrix} & w.p. \frac{3}{4} \\ f_2 : \begin{bmatrix} x' & \mapsto x \\ y' & \mapsto y \\ \mathit{count}' & \mapsto \mathit{count} + 1 \end{bmatrix} & w.p. \frac{1}{4} \end{cases}$$

For $\text{pre}\mathbb{E}_{\tau_1}(e)$ it follows:

$$\begin{aligned}
\text{pre}\mathbb{E}_{\tau_1}(1+2x-3y) &= \sum_{j=1}^{2} p_j \cdot \mathbb{E}_{\mathcal{D}_R}(\text{pre}(1+2x-3y,\ f_j) \mid \boldsymbol{x}) \\
&= \frac{3}{4} \cdot \mathbb{E}_{\mathcal{D}_R}(1+2\cdot(x+r_1)-3\cdot(y+2)) + \frac{1}{4}\mathbb{E}_{\mathcal{D}_R}\cdot(1+2x-3y) \\
&= \frac{3}{4} \cdot (1+2\cdot\mathbb{E}_{r_1}(x+r_1)-3\cdot(y+2)) + \frac{1}{4}\cdot(1+2x-3y) \\
&= -\frac{7}{2}+2x-3y+\frac{3}{2}\cdot\mathbb{E}_{r_1}(r_1)
\end{aligned}$$

We remember from the program code that r_1 was drawn from the uniform distribution $[0,2]$, thus the expected value $\mathbb{E}_{r_1}(r_1)=1$. Using this we obtain

$$\text{pre}\mathbb{E}_{\tau_1}(1+2x-3y) = -2+2x-3y$$

■

Now we will use these definitions of pre-expectations with respect to a transition to formulate pre-expectations over all transitions inside a loop.

Definition 2.6 (Pre-expectation over all transitions)**.** Let e' be an affine expression involving the program values $\boldsymbol{x}$. Then:

$$\text{pre}\mathbb{E}(e') = \sum_{\tau_i\in\mathcal{T}} \mathbb{1}_{\mathbf{g}_{\tau_i}}(\boldsymbol{x}) \cdot \text{pre}\mathbb{E}_{\tau_i}(e')$$

where $\mathbb{1}_g(\boldsymbol{x})$ is the indicator function:

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

Given that the current state is $(\boldsymbol{x}_n, n)$ we now want to find out the expectation of an expression e over the distribution of all possible successor states $(\boldsymbol{x}_{n+1}, n+1)$. The following lemma follows directly from the definition of the pre-expectation operator.

Lemma 2.1. The expected value of e over the post-state distribution starting from state $(\boldsymbol{x}_n, n)$ is the value of the pre-expectation $\text{pre}\mathbb{E}(e')$ evaluated over the current state $\boldsymbol{x}_n$:

$$\mathbb{E}(e(\boldsymbol{x}_{n+1}) \mid \boldsymbol{x}_n, n) = \text{pre}\mathbb{E}(e') = \sum_{\tau_i\in\mathcal{T}} \mathbb{1}_{\mathbf{g}_{\tau_i}}(\boldsymbol{x}) \cdot \text{pre}\mathbb{E}_{\tau_i}(e')$$

We can now extend this lemma to the full distribution $\mathcal{D}_n$ from which $\boldsymbol{x}_n$ is drawn.

Lemma 2.2. Let e be an affine program expression. Then:

$$\mathbb{E}_{\mathcal{D}_{n+1}}(e) = \mathbb{E}_{\mathcal{D}_n}(\text{pre}\mathbb{E}(e')) = \mathbb{E}_{\mathcal{D}_n}\left(\sum_{\tau_i\in\mathcal{T}} \mathbb{1}_{\mathbf{g}_{\tau_i}}(\boldsymbol{x}) \cdot \text{pre}\mathbb{E}_{\tau_i}(e')\right)$$

3 Expectation Invariants

Expectation invariants are inequalities on the expected values of expressions on each loop iteration. These can be used to prove properties of a loop. We are now going to define expectation invariants and in the next chapter we will show how we can compute fixed points of expectation invariants

3.1 Expectation Invariants

Let $\langle \mathcal{T}, \mathcal{D}_0, n \rangle$ be a probabilistic loop. We define the expectation of an affine expression e at iteration n as $\mathbb{E}(e \mid n) = \mathbb{E}_{\mathcal{D}_n}(e)$, where $\mathcal{D}_n$ denotes the state distribution after n iterations.

Definition 3.1 (Expectation invariants)**.** An affine expression e over the program variables X is called an expectation invariant (EI) if and only if $\mathbb{E}(e \mid n) \geq 0$ for all $n \geq 0$.

In other words, expectation invariants are exactly those expressions for which the expected values are non-negative over the initial distribution and remain non-negative for all iterations of the probabilistic loop.

Example: We again consider the program from Figure 3 and show that the expression $e = 2y - x$ is an expectation invariant. Initially it holds $\mathbb{E}(2y - x \mid 0) = \mathbb{E}_{\mathcal{D}_0}(2y - x) = 2\mathbb{E}_{\mathcal{D}_0}(y) - \mathbb{E}_{\mathcal{D}_0}(x) = 2 - (-1) = 3 \geq 0$. Further, $\mathbb{E}(2y - x \mid i) = 2\mathbb{E}(y \mid i) - \mathbb{E}(x \mid i) \geq 0$ holds at any step i the expected value of y grows faster than the expected value of x. Therefore, e is an expectation invariant of $\mathcal{P}$.

■

We are now going to determine whether an expression e is an expectation invariant, i.e. show that the expected value is non-negative for all steps. To do this we are going to approximate the distribution $\mathcal{D}_n$ for all n. Alternatively one could find an argument based on mathematical induction to prove that an expression e is an EI.

Definition 3.2 (Admissible distribution)**.** A distribution $\mathcal{D}$ over the set of program variables X is admissible if and only if $\mathbb{E}_{\mathcal{D}}(p(\boldsymbol{x}))$ exists and is finite for any polynomial $p(x)$ over the program variables.

We assume that for any program $\mathcal{P}$ that we attempt to analyse, $\mathcal{D}_0$ is admissible, and for each transition τ, the distribution of random variables $\mathcal{D}_R$ is admissible. Under these assumption it can be shown that the following holds [2].

Lemma 3.1. For all $n \in \mathbb{N}$, the distribution $\mathcal{D}_n$ is admissible.

In practice the explicit construction of $\mathcal{D}_n$ for each n is quite expensive. Thus, we will use the principle of inductive expectation invariants.

Definition 3.3 (Inductive expectation invariants)**.** Let $E = \{e_1, \ldots, e_m\}$ be a set of expressions where each e_i is a linear or polynomial expression involving the program variables. Then the set E forms an inductive expectation invariant of the program $\mathcal{P}$ if and only if for each $e_j,\ j \in [1, m]$, the following holds:

1. $\mathbb{E}_{\mathcal{D}_0}(e_j) \geq 0$, i.e., the expectation at the initial step is non-negative.
2. For every admissible distribution $\mathcal{D}$ over the set of program variables X,

$$(\mathbb{E}_{\mathcal{D}}(e_1) \geq 0 \wedge \ldots \wedge \mathbb{E}_{\mathcal{D}}(e_m) \geq 0) \Rightarrow \mathbb{E}_{\mathcal{D}}(\text{pre}\mathbb{E}(e_j)) \geq 0$$

This way of defining inductive expectation invariants follows the Floyd-Hoare approach to abstract away the distribution at the n-th step by usage of the inductive invariant itself to show that the invariant continues to hold in the next step. As we are using any admissible distribution $\mathcal{D}$ we are also not bound to a specific $\mathcal{D}_n$. It can be shown that the following theorem holds.

Theorem 3.1. Let $E : \{e_1, \ldots, e_m\}$ be an inductive expectation invariant. It follows that each $e_j \in E$ is an expectation invariant.

In practice, Definition 3.3 is unhandy, as the quantification over all possible admissible distributions $\mathcal{D}$ over the set of program variables X is a higher-order quantifier. Due to the fact that reasoning with this quantifier is not practical, we are going to refine the approach. This will be done by applying the facts from the following theorem [2].

Theorem 3.2 (Facts about expectations over admissible distributions)**.** The following facts hold for all admissible distributions $\mathcal{D}$ over a σ-algebra $\mathcal{X}$, linear assertion $\varphi : \bigwedge_{i=1}^{j} \boldsymbol{a}_i^T \cdot \boldsymbol{y} \leq b$, $a_i, y \in \mathcal{R}^n$, $b \in \mathbb{R}$ and linear or polynomial expressions $e, e_1, \ldots, e_k$.

1. Linearity of expectations: $\mathbb{E}_{\mathcal{D}}(\lambda_1 e_1 + \ldots + \lambda_k e_k) = \lambda_1 \mathbb{E}_{\mathcal{D}}(e_1) + \ldots + \lambda_k \mathbb{E}_{\mathcal{D}}(e_k)$, for all $\lambda_i \in \mathbb{R}$
2. Indicator functions: $\mathbb{E}_{\mathcal{D}}(\mathbb{1}_{e \geq 0} \times e) \geq 0$, and in general, if $\varphi \models e \geq 0$ then $\mathbb{E}_{\mathcal{D}}(\mathbb{1}_{\varphi} \times e) \geq 0$, provided that $[\![\varphi]\!]$ is measurable
3. $\mathbb{E}_{\mathcal{D}}(\mathbb{1}_{\varphi} e + \mathbb{1}_{\neg\varphi} e) = \mathbb{E}_{\mathcal{D}}(e)$, provided that $[\![\varphi]\!]$ is measurable

where $[\![\varphi]\!]$ means the set of solutions for φ.

The central idea now is to use the facts from the theorem and reformulate the second part of Definition 3.3 to a simple quantified statement in first-order linear arithmetic. To do so, we express the pre-expectation $\text{pre}\mathbb{E}(e_j)$ for each $e_j \in E$ as:

$$\text{pre}\mathbb{E}(e_j) = \sum_{i=1}^{m} \boldsymbol{\lambda}_{j,i} e_i + \sum_{p} \boldsymbol{\mu}_{j,p} \times (\mathbb{1}_{\varphi_p} g_p)$$

where $\boldsymbol{\lambda}_{j,i} \geq 0$ and $\boldsymbol{\mu}_{j,p} \geq 0$ are real-valued multipliers, g_p is a boolean expression over the program variables and φ_p is an assertion such that $\varphi_p \models g_p \geq 0$. The origin of the expressions g_p and assertion φ_p will be made clear shortly. It can be shown that this way of computing $\text{pre}\mathbb{E}(e_j)$ can be derived from Definition 2.6 by reformulation. By usage of Theorem 3.2 it can be shown that the following holds [2].

Lemma 3.2. Let $E = \{e_1, \ldots, e_m\}$ be a finite set of expressions. If $\text{pre}\mathbb{E}(e_j) \geq 0$ for all e_j (with $\text{pre}\mathbb{E}(e_j)$ as defined above), then E satisfies condition 2. of Definition 3.3.

3.2 Conic Inductive Expectation Invariants

Let $\mathcal{P}$ be a probabilistic program with transitions $\tau_1, \ldots, \tau_m$. 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 boolean expressions. Let $\mathbf{g}_i : (g_{i,1} \ldots g_{i,n_i})^T$ be a vector representing $\mathbf{g}_i$. Further, let $E = \{e_1, \ldots, e_m\}$ be a finite set of expressions, we denote the vector representing these expressions as $\mathbf{e} : (e_1, \ldots, e_m)^T$.

Definition 3.4 (Conic inductive expectation invariants)**.** The finite set $E = \{e_1, \ldots, e_m\}$ is a conic inductive invariant of the program $\mathcal{P}$ if and only if for each $e_j \in E$ the following holds:

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

$$\exists \boldsymbol{\lambda}_j \geq 0, \forall \tau_l \in \mathcal{T}, \exists \boldsymbol{\mu}_l \geq 0 \left(\text{pre}\mathbb{E}_{\tau_l}(e_j) = \boldsymbol{\lambda}_j^T e + \boldsymbol{\mu}_l^T \mathbf{g}_l\right)$$

Note that the order of quantifications in this equation is important to ensure that the first part of Theorem 3.2 is applicable.

Example: Consider the program from Figure 3 (right) again. The set $E = \{e_1 : y - 2x, e_2 : -2x + y - 3, e_3 : 2x - y + 3\}$ is a conic inductive invariant for the program. Consider e_1. We have:

$$\text{pre}\mathbb{E}_{\tau_1}(e_1) = \mathbb{E}_{r_1}\left(\frac{3}{4}(-2(x + r_1) + (y + 2) + \frac{1}{4}(-2x + y)\right) = y - 2x$$

Likewise, $\text{pre}\mathbb{E}_{\tau_2}(e_1) = e_1$, as τ_2 is a stuttering transition. Setting $\boldsymbol{\lambda} = (1, 0, 0)^T$ and $\boldsymbol{\mu} = \mathbf{0}$ yields $\text{pre}\mathbb{E}(e_1) = \boldsymbol{\lambda}^T \boldsymbol{e} + \boldsymbol{\mu}^T \times \mathbb{1}_{x+y \leq 10}$
It can be shown that $\text{pre}\mathbb{E}_{\tau_1}(e_2)$ and $\text{pre}\mathbb{E}_{\tau_1}(e_3)$ can be expressed in a similar way, thus E is a conic inductive expectation invariant.

■

3.3 Pre-Expectation of Closed Cones

Until now we only reasoned about finite sets of expressions $E = \{e_1, \dots, e_k\}$, satisfying the conditions in definition 3.3. We are now going to transfer the notion from a finite set of expressions to a finitely generated cone of these expressions and later on use these cones for the fix point characterization given in the next section.

Definition 3.5 (Cones)**.** Let $E = \{e_1, \dots, e_k\}$ be a finite set of affine program expressions over the program variables $\boldsymbol{x}$. The set of conic combinations (the finitely generated cone) of E is defined as

$$\text{Cone}(E) = \left\{ \lambda_0 + \sum_{i=1}^{k} \lambda_i e_i \mid \lambda_i \in \mathbb{R}^+,\ 0 \le i \le k \right\}$$

Expressions e_j are called the generators of the cone.

For a non-empty linear assertion $\varphi : \bigwedge_{i=1}^{k} \mathbb{E}_{\mathcal{D}_n}(e_i) \ge 0$ and any $n \ge 0$, it is easy to show that $\varphi \models \mathbb{E}_{\mathcal{D}_n}(e) \ge 0$ if and only if $e \in \text{Cone}(e_1, \dots, e_k)$. Moreover, if E is an inductive expectation invariant, then $e \in \text{Cone}(E)$ is an expectation invariant of the program $\mathcal{P}$.

Example: Consider the finite set of expressions $E = \{e_1 : y - x, e_2 : 2y + \mathit{count}\}$ over the program from Figure 3 (right) and take without proof e_1, e_2 are EI. Now consider $e = 4y - 2x + \mathit{count} = 2 \cdot e_1 + e_2$. $e \in \text{Cone}(E)$ as it is a linear combination of elements of the generators of the cone. $\mathbb{E}_{\mathcal{D}_n}(e) = 4 \cdot \mathbb{E}_{\mathcal{D}_n}(y) - 2 \cdot \mathbb{E}_{\mathcal{D}_n}(x) + \mathbb{E}_{\mathcal{D}_n}(\mathit{count}) \ge 0$ as $\mathbb{E}_{\mathcal{D}_n}(\mathit{count}) = n$, and for all n $\mathbb{E}_{\mathcal{D}_n}(y) \ge \mathbb{E}_{\mathcal{D}_n}(x)$ explained in a previous example. Therefore e is also an expectation invariant of the program.

■

4 Expectation Invariants as Fixed Points

In the following we will show that the notion of conic inductive expectation invariants can be expressed as a fixed point of a monotone operator over finitely generated cones. This monotone operator will allow us to start from the cone that represents all expressions and perform iterations until convergence. As this computation might take a lot of iterations (up to infinitely many), we are going to define a dualized widening operator to ensure a fast convergence. With usage of this dualized widening operator the iteration is guaranteed to converge to the fixed point in finitely many steps.

In the following let $\mathcal{P}$ be a probabilistic program over variables $\boldsymbol{x}$ with transitions $\mathcal{T} : \{\tau_1, \dots, \tau_k\}$ and initial distribution $\mathcal{D}_0$. Further, we will describe affine expressions of the form $c_0 + \boldsymbol{c}^T\boldsymbol{x}$ for $c_0 \in \mathbb{R}$, $\boldsymbol{c} \in \mathbb{R}^m$. Let $\mathbb{A}(\boldsymbol{x})$ be the set of all affine expressions over $\boldsymbol{x}$.

Example: Consider the expression $e : 2x + 5y - 3$. This can be written as:

$$e : (2, 5) \times \begin{pmatrix} x \\ y \end{pmatrix} - 3$$

■

As stated above, the approach uses finitely generated cones $I = \text{Cone}(E)$ where $E = \{e_1, \cdots, e_k\}$ is a finite set of affine expressions over $\boldsymbol{x}$. Once a fixed point is found by the technique explained below, we obtain a cone $I^* : \text{Cone}(E^*)$, wherein E^* is a conic inductive invariant according to definition 3.4.

A finitely generated cone of affine expressions $I = \text{Cone}(E)$ can be represented by a polyhedral cone of its coefficients $C(I) : \{(c_0, \boldsymbol{c}) \mid c_0 + \boldsymbol{c}^T\boldsymbol{x} \in I\}$, where the generators of $C(I)$ are coefficient vectors $(c_{0,i}, \boldsymbol{c}_i)$ representing the expression $e_i : c_{0,i} + \boldsymbol{c}_i^T\boldsymbol{x}$.
We now lift the notion of the pre-expectation operator $\text{pre}\mathbb{E}_\tau$ from expressions to cones.

Definition 4.1 (Pre-expectations over cones)**.** Let $E = \{e_1, \ldots, e_m\}$ be a set of affine expressions, and let $\tau : \langle \mathbf{g}, \mathcal{F} \rangle$ be a transition, wherein $\mathbf{g} : \bigwedge_{l=1}^{p} g_l \geq 0$.
The pre-expectation of a cone $I : \text{Cone}(E)$ with respect to τ is defined as:

$$\text{pre}\mathbb{E}_\tau(I) = \{(e, \lambda) \in \mathbb{A}(x) \times \mathbb{R}^m \mid \lambda \geq 0 \wedge \exists \mu \geq 0 (\text{pre}\mathbb{E}_\tau(e) \equiv \sum_{j=1}^{m} \lambda_j e_j + \sum_{i=1}^{p} \mu_i g_i)\}$$

The set $\text{pre}\mathbb{E}_\tau(I)$ contains all affine program expressions e, such that e belongs to the conic hull of I and the cone generated by the guard assertion. The parameter λ can be seen as a certificate, which is attached to each expression to show its membership back in the cone. This is done to ensure the proper order of quantifications in definition 3.4.

Lemma 4.1. For every cone C, $\text{pre}\mathbb{E}_\tau(C)$ is a cone.

For a more detailed explanation of how to compute the pre-expectation of a cone as well as the fixed point computation across multiple transitions we refer to [2].

Now, we will look at the pre-expectation operator not only for single transitions but over all transitions at once.

Definition 4.2. Let I be a finitely generated cone of affine expressions. The pre-expectation over all transitions in $\mathcal{T} = \{\tau_1, \ldots, \tau_k\}$ can be computed as:

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

An expression e belongs to $\mathrm{pre}\mathbb{E}(I)$ if for some $\lambda \geq 0$ $(e, \lambda) \in \mathrm{pre}\mathbb{E}_{\tau_j}(I)$ for each transition $\tau_j \in \mathcal{T}$.
For a cone $C(I)$, we first compute the cones $C(\hat{I}_1), \dots C(\hat{I}_k)$ with $C(\hat{I}_i) : \mathrm{pre}\mathbb{E}_{\tau_i}(I)$ for all $\tau_1, \dots, \tau_k \in \mathcal{T}$ respectively. We then compute $C(I') : (\exists \lambda \geq 0) \bigcup_{j=1}^{k} C(\hat{I}_j)$ representing $I' : \mathrm{pre}\mathbb{E}(I)$, by intersecting the cones $C(\hat{I}_j)$ and projecting the dimensions corresponding to λ.

For simplicity, we assume that $\mathbb{E}_{\mathcal{D}_0}(x)$ is computable and $\mathcal{D}_0$ is known. The initial cone for the iteration steps is given as

$$I_0 : \mathrm{Cone}(\{1, x_1 - \mathbb{E}_{\mathcal{D}_0}(x_1), \mathbb{E}_{\mathcal{D}_0}(x_1) - x_1, \dots, x_n - \mathbb{E}_{\mathcal{D}_0}(x_n), \mathbb{E}_{\mathcal{D}_0}(x_n) - x_n\})$$

where the cone I_0 represents the invariant candidates $x_i = \mathbb{E}_{\mathcal{D}_0}(x_i)$.

Example: The initial cone for the probabilistic program from Figure 3 is:

$$\begin{aligned} I_0 :\ & \mathrm{Cone}(\{1, x - \mathbb{E}_{\mathcal{D}_0}(x), \mathbb{E}_{\mathcal{D}_0}(x) - x, \mathbb{E}_{\mathcal{D}_0}(y) - y, y - \mathbb{E}_{\mathcal{D}_0}(y), \\ & \qquad \mathbb{E}_{\mathcal{D}_0}(\mathit{count}) - \mathit{count}, \mathit{count} - \mathbb{E}_{\mathcal{D}_0}(\mathit{count})\}) \\ =\ & \mathrm{Cone}(\{1, x+1, y-1, 1-y, \mathit{count}, -\mathit{count}\}) \end{aligned}$$

■

Definition 4.3. Let $\mathcal{G}(I) = I_0 \bigcap \mathrm{pre}\mathbb{E}(I)$, where I_0 is the initial cone.

Example: We again consider the program in Figure 3 (right), I_0 as in the previous example. It follows

$$\begin{aligned} \mathrm{pre}\mathbb{E}_{\tau_1}(I_0) = \{ & (1, (1,0,0,0,0,0)^T), (x+1, (0.75, 1, 0, 0, 0, 0)^T), \\ & (y-1, (2.5, 0, 1, 0, 0, 0)^T), (1-y, (1.5, 0, 0, 1, 0, 0)^T), \\ & (\mathit{count}, (1, 0, 0, 0, 1, 0)^T)\} \end{aligned}$$

$\mathrm{pre}\mathbb{E}_{\tau_2}$ is trivially obtained by attaching certificates to each expression that correspond to the identity of the expression. We now compute $\mathrm{pre}\mathbb{E}(I_0) = \mathrm{pre}\mathbb{E}_{\tau_1}$. Further,

$$I_1 = \mathcal{G}(I_0) = I_0 \bigcap \mathrm{pre}\mathbb{E}(I_0) = \{1, x+1, y-1, 1-y, \mathit{count}\}$$

It can be shown that $I_1 = \mathcal{G}(I_1)$ thus I_1 represents the cone containing only fixed points of the probabilistic loop.

■

Theorem 4.1. The operator $\mathcal{G}$ satisfies the following properties:

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

We can now use this theorem to compute the greatest fixed point of $\mathcal{G}$, which represents the largest cone of expressions whose generators satisfy definition 3.4. This can be implemented by iterating until a pre-fixed point is obtained which, in the ideal case, is also the greatest fixed point of $\mathcal{G}$ [1]. In each step it holds

$$(J_0 : \mathbb{A}(x)) \supseteq (J_1 : \mathcal{G}(J_0)) \supseteq \ldots \supseteq (J_2 : \mathcal{G}(J_1))$$

Note that there might be chains that are infinite, e.g., alternating between two elements of the initial cone, and therefore it is possible that the greatest fixed point can not be found in finitely many steps by the Kleene iteration. To force the downward iteration to converge we will use a dual widening operator.

Definition 4.4 (Dual widening)**.** Let I_1, $I_2 = \mathcal{G}(I_1)$ 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 sequence $J'_0 = J_0, J'_n = \widetilde{\nabla}(J'_{n-1}, J_n)$ converges in finitely many steps.

$I' = \widetilde{\nabla}(I_1, I_2)$ again is a cone, thus it can be used for further computations.

Definition 4.5 (Standard dual widening)**.** Let $I_1 = \text{Cone}(e_1, \ldots, e_k)$ and $I_2 = \text{Cone}(g_1, \ldots, g_k)$ 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 $I = \text{Cone}(g_i \mid e_i \in I_2)$. Cone I is the cone generated by generators of I_1 that are also in I_2

It can be shown that the standard dual widening is a dual widening following Definition 4.4.

A popular method to compute an approximation of the greatest fixed point when using dual widening is to delay widening for a fixed number of iterations. This way the inaccuracy can be greatly reduced to fulfill the requirements.

5 Experimental Results

The table below shows a summary of the results from the experiments of [2]: $|X|$ is the number of program variables, $|\mathcal{T}|$ the number of transitions, $\#$ the number of iterations to

Name	$\lvert X\rvert$	$\lvert\mathcal{T}\rvert$	#	$\widetilde{\nabla}$	Time
MOT-EXAMPLE	3	2	2	No	$\leq \varepsilon$
MOT-EX-LOOP-INV	3	2	2	No	0.10
MOT-EX-POLY	9	2	2	No	0.18
2D-WALK	4	4	7	Yes	$\leq \varepsilon$
AGGREGATE-RV	3	2	2	No	$\leq \varepsilon$
HARE-TURTLE	3	2	2	No	$\leq \varepsilon$
COUPON5	2	5	2	No	$\leq \varepsilon$
HAWK-DOVE-FAIR	6	2	2	No	$\leq \varepsilon$
HAWK-DOVE-BIAS	6	2	2	No	$\leq \varepsilon$
FAULTY-INCR	2	2	7	Yes	$\leq \varepsilon$

convergence, $\widetilde{\nabla}$ - an indicator if the dual widening was used and time (in seconds) the time needed to compute the fixed points, where all runs that took less than $\varepsilon = 0.05s$ are not monitored.
All examples except MOT-EX-LOOP-INV and MOT-EX-POLY run in under 0.05 seconds. The usage of the widening operator was delayed by at least 5 iterations to force convergence only if necessary.

6 Conclusion

We have shown how pre-expectations over single transitions of a probabilistic loop can be computed for affine expressions as well as for sets of affine expressions. Further, we showed how to compute pre-expectations over all possible transitions and how to compute pre-expectations of finitely generated sets of expressions. We then used these pre-expectations to formulated an operator that can be applied iteratively and yields a set of fixed points for the expectation invariants once it converges. Then we introduced the widening operator to ensure the convergence of the iterations. The experimental results show that the invariants for most of the tests could be found in less than 3 iterations and the computation was quite fast.

References

[1] A. Chakarov and S. Sankaranarayanan. Expectation invariants for probabilistic program loops as fixed points. In Static Analysis - 21st International Symposium, SAS 2014, Munich, Germany, September 11-13, 2014. Proceedings, pages 85–100, 2014.

[2] A. Chakarov and S. Sankaranarayanan. Expectation invariants for probabilistic program loops as fixed points (extended version). Technical report, University of Colerado, 2014.

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

[4] P. Mardziel, S. Magill, M. Hicks, and M. Srivatsa. Dynamic enforcement of knowledge-based security policies. In Proceedings of the 24th IEEE Computer Security Foundations Symposium, CSF 2011, Cernay-la-Ville, France, 27-29 June, 2011, pages 114–128, 2011.

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