Expectation Invariants for Probabilistic Program Loops as Fixed Points

Phillip Florian

February 3, 2015
Introduction

- Probabilistic programs use random variables
  ⇒ Properties of probabilistic programs are hard to verify
Introduction

- Probabilistic programs use random variables
  ⇒ Properties of probabilistic programs are hard to verify

- Expectation invariants are expressions over the program variables that stay non-negative all the time
  ⇒ They can be used to verify properties
Introduction

- Probabilistic programs use random variables
  ⇒ Properties of probabilistic programs are hard to verify

- Expectation invariants are expressions over the program variables that stay non-negative all the time
  ⇒ They can be used to verify properties

- Compute expectation invariants as fixed points
Introduction

- Probabilistic programs use random variables
  ⇒ Properties of probabilistic programs are hard to verify

- Expectation invariants are expressions over the program variables that stay non-negative all the time
  ⇒ They can be used to verify properties

- Compute expectation invariants as fixed points
  - The presented algorithm computes a set of expectation invariants
  - Only works with some restrictions to the program
Probabilistic programs

The following notion will be used:

- $\mathcal{P}$: A probabilistic program
- $X = \{x_1, \ldots, x_m\}$: A finite set of program variables
- $R = \{r_1, \ldots, r_l\}$: A finite set of random variables
- $\mathcal{D}_R$: The joint distribution of random variables $R$
- $x, r$: The vectors denoting the valuation of all program and random variables respectively
Probabilistic loops

Definition (Probabilistic loops)

A probabilistic loop of $P$ is a tuple $\Tuple{\mathcal{T}, \mathcal{D}_0, n}$, with

- $\mathcal{T} : \{\tau_1, \ldots, \tau_k\}$: A finite set of probabilistic transitions
- $\mathcal{D}_0$: The initial probability distribution of the program variables
- $n$: A loop counter
Probabilistic loops

Definition (Probabilistic loops)

A probabilistic loop of $\mathcal{P}$ is a tuple $\langle \mathcal{T}, D_0, n \rangle$, with

- $\mathcal{T} : \{\tau_1, \ldots, \tau_k\}$: A finite set of probabilistic transitions
- $D_0$: The initial probability distribution of the program variables
- $n$: A loop counter

A probabilistic transition $\tau_i : \langle g_i, F_i \rangle$ consists of

- A guard $g_i(x)$ over $X$
- An update function $F_i(x, r)$ s.t. after taking the transition it holds: $x' = F_i(x, r)$. 
Example

```c
int x := rand (0,2)
while (x<=10){
    x:= x + rand (0,2)
}
```
Example

```c
int x := rand (0,2)
while (x <= 10) {
    x := x + rand (0,2)
}
```

Can be expressed as $\langle \mathcal{T}, \mathcal{D}_0, n \rangle$

- $\mathcal{T} = \{\tau_1\}$
Example

```c
int x := rand (0,2)
while (x<=10){
    x:= x + rand (0,2)
}
```

Can be expressed as $\langle \mathcal{T}, \mathcal{D}_0, n \rangle$

- $\mathcal{T} = \{\tau_1\}$
- $g_1(x) = x \leq 10$
Example

```c
int x := rand(0,2)
while (x <= 10) {
    x := x + rand(0,2)
}
```

Can be expressed as $\langle \mathcal{T}, \mathcal{D}_0, n \rangle$

- $\mathcal{T} = \{\tau_1\}$
- $g_1(x) = x \leq 10$
- $\mathcal{F}_1(x,r) = x + r_1$
- $r_1 = U(0,2)$
Example

```c
int x := rand (0,2)
while (x <= 10){
    x := x + rand (0,2)
}
```

Can be expressed as $\langle \mathcal{T}, D_0, n \rangle$

- $\mathcal{T} = \{ \tau_1 \}$
  - $g_1(x) = x \leq 10$
  - $\mathcal{F}_1(x, r) = x + r_1$
    - $r_1 = U(0,2)$

- $D_0 : \langle x \rangle = U[0,2]$
Example

```c
int x := rand (0,2)
while (x <= 10) {
    x := x + rand (0,2)
}
```

Can be expressed as \( \langle \mathcal{T}, D_0, n \rangle \)

- \( \mathcal{T} = \{\tau_1\} \)
- \( g_1(x) = x \leq 10 \)
- \( \mathcal{F}_1(x,r) = x + r_1 \)
  - \( r_1 = U(0,2) \)
- \( D_0 : \langle x \rangle = U[0,2] \)
- \( n = 0 \)
Definition (Piecewise linear transitions)

\( \tau : \langle g, F(x, r) \rangle \) is a piecewise linear transition if:

- \( g \) is a linear guard over \( X \)
- \( F(x, r) \) is a piecewise linear function and may be written as:
  \[
  F(x, r) = \begin{cases} 
  f_1 : A_1 + B_1 + d_1, & \text{with probability } p_1, \\
  \vdots \\
  f_k : A_k + B_k + d_k, & \text{with probability } p_k.
  \end{cases}
  \]

- \( A_i, B_i, d_i \) are used to model the changes to the program variables occurring in the loop.

P. Florian

Expectation Invariants as Fixed Points

February 3, 2015 7 / 33
### Definition (Piecewise linear transitions)

\( \tau : \langle g, F(x, r) \rangle \) is a **piecewise linear transition** if:

- \( g \) is a *linear guard* over \( X \)
Piecewise linear transitions

**Definition (Piecewise linear transitions)**

\( \tau : \{g, F(x, r)\} \) is a piecewise linear transition if:

- \( g \) is a linear guard over \( X \)
- \( F(x, r) \) is a piecewise linear function and may be written as:

\[
F(x, r) = \begin{cases} 
  f_1 : A_1 x + B_1 r + d_1, & \text{with probability } p_1 \\
  \vdots \\
  f_k : A_k x + B_k r + d_k, & \text{with probability } p_k 
\end{cases}
\]
Piecewise linear transitions

Definition (Piecewise linear transitions)

τ : ⟨g, F(x, r)⟩ is a piecewise linear transition if:

- g is a linear guard over X
- F(x, r) is a piecewise linear function and may be written as:

\[
F(x, r) = \begin{cases} 
  f_1 : A_1 x + B_1 r + d_1, \quad \text{with probability } p_1 \\
  \vdots \\
  f_k : A_k x + B_k r + d_k, \quad \text{with probability } p_k 
\end{cases}
\]

- \(f_1, \ldots, f_k\): Identifier for different outcomes of Bernoulli choices
Piecewise linear transitions

Definition (Piecewise linear transitions)

\( \tau : (g, F(x,r)) \) is a piecewise linear transition if:

- \( g \) is a linear guard over \( X \)
- \( F(x,r) \) is a piecewise linear function and may be written as:

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

- \( f_1, \ldots, f_k \): Identifier for different outcomes of Bernoulli choices
- \( p_1, \ldots, p_k \): Probabilities for choosing the corresponding fork
**Definition (Piecewise linear transitions)**

$\tau : \langle g, \mathcal{F}(x, r) \rangle$ is a piecewise linear transition if:

- $g$ is a linear guard over $X$
- $\mathcal{F}(x, r)$ is a piecewise linear function and may be written as:

$$\mathcal{F}(x, r) = \begin{cases} f_1 : A_1x + B_1r + d_1, & \text{with probability } p_1 \\ \vdots \\ f_k : A_kx + B_kr + d_k, & \text{with probability } p_k \end{cases}$$

- $f_1, \ldots, f_k$: Identifier for different outcomes of Bernoulli choices
- $p_1, \ldots, p_k$: Probabilities for choosing the corresponding fork
- $A_i \in \mathbb{R}^{m \times m}, B_i \in \mathbb{R}^{m \times l}, d_i \in \mathbb{R}^m$ are used to model the changes to the program variables occurring in the loop.
Example

```c
int x := rand (-5,3)
int y := rand (-3,5)
int c := 0
while (true){
    if (x+y<=$10)
        if flip (3/4)
            x:= x + rand (0,2)
            y:= y + 2
            c++
        else
            do nothing
    }
```
Example (continued)

The corresponding piecewise linear transition \( \tau : \langle g, \mathcal{F} \rangle \)

- \( g(x) = x + y \leq 10 \)
Example (continued)

The corresponding piecewise linear transition \( \tau : \langle g, F \rangle \)

- \( g(x) = x + y \leq 10 \)

- \( F(x, r) = \)
Example (continued)

The corresponding piecewise linear transition $\tau : \langle g, F \rangle$

- $g(x) = x + y \leq 10$

- $f_1 : \begin{pmatrix} x \\ y \\ c \end{pmatrix} + \begin{pmatrix} r_1 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 2 \\ 1 \end{pmatrix}$, $p_1 = \frac{3}{4}$

- $F(x, r) =$
Example (continued)

The corresponding piecewise linear transition $\tau : \langle g, F \rangle$

- $g(x) = x + y \leq 10$
- $F(x, r) = \begin{cases} f_1 : \begin{pmatrix} x \\ y \\ c \end{pmatrix} + \begin{pmatrix} r_1 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 2 \\ 1 \end{pmatrix}, & p_1 = \frac{3}{4} \\ f_2 : \begin{pmatrix} x \\ y \\ c \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, & p_2 = \frac{1}{4} \end{cases}$
Restrictions for this approach

- Probabilistic loops are not nested.
- For simplicity, for all nested loops there are equivalent unnested variants.
- All transitions are piecewise linear.
- Exactly one transition can be taken in every iteration.
- All expressions $e_i$ are linear expressions:
  
  $$e_i = c_0 + \sum_{m=0}^{i} P_{m} \cdot x_i,$$

  where $c_0$ and $P_i$ are constants.
Restrictions for this approach

- Probabilistic loops are not nested
Probabilistic Loops

Restrictions for this approach

- Probabilistic loops are not nested
  - For simplicity
  - For all nested loops there are equivalent unnested variants

All transitions are piecewise linear
Exactly one transition can be taken in every iteration
All expressions $e(x)$ are linear expressions

$e(x) = c_0 + \sum_{i=0}^{m} P_i x_i$, $c_0$, $i \in R$
Restrictions for this approach

- Probabilistic loops are not nested
  - For simplicity
  - For all nested loops there are equivalent unnested variants

- All transitions are piecewise linear
Restrictions for this approach

- Probabilistic loops are not nested
  - For simplicity
  - For all nested loops there are equivalent unnested variants

- All transitions are piecewise linear

- Exactly one transition can be taken in every iteration
  - The loop might need to be modified

All expressions $e()$ are linear expressions:

$$e() = c_0 + P_{m_i=0} i \cdot x_i,$$

$c_0$, $i_2 R P.$
Probabilistic Loops

Restrictions for this approach

- Probabilistic loops are not nested
  - For simplicity
  - For all nested loops there are equivalent unnested variants

- All transitions are piecewise linear

- Exactly one transition can be taken in every iteration
  - The loop might need to be modified

- All expressions $e(x)$ are linear expressions
  - $e(x) = c_0 + \sum_{i=0}^{m} \lambda_i \cdot x_i$, $c_0, \lambda_i \in \mathbb{R}$
Execution model

To model an execution of a probabilistic loop we use tuples \((x_n, n)\) as states.
Execution model

To model an execution of a probabilistic loop we use tuples \((x_n, n)\) as states, where:

- \(x_n\) represents the program variables at loop-iteration \(n\)
Execution model

To model an execution of a probabilistic loop we use tuples \((x_n, n)\) as states, where:

- \(x_n\) represents the program variables at loop-iteration \(n\)
- \((x_0, 0)\) is an \textit{initial state} if \(x_0\) is drawn from \(D_0\)
Execution model

To model an execution of a probabilistic loop we use tuples \((x_n, n)\) as states, where:

- \(x_n\) represents the program variables at loop-iteration \(n\)
- \((x_0, 0)\) is an \textit{initial state} if \(x_0\) is drawn from \(D_0\)
- \((x_i, i)\) is \textit{predecessor} of \((x_{i+1}, i + 1)\) if for a transition \(\tau : \langle g, F(x, r) \rangle\)
  - \(x_i \models g\)
  - \(\exists r \in D_R, x_{i+1} = F(x_i, r)\)
Execution model

To model an execution of a probabilistic loop we use tuples \((x_n, n)\) as states, where:

- \(x_n\) represents the program variables at loop-iteration \(n\)
- \((x_0, 0)\) is an initial state if \(x_0\) is drawn from \(\mathcal{D}_0\)
- \((x_i, i)\) is predecessor of \((x_{i+1}, i + 1)\) if for a transition \(\tau : (g, \mathcal{F}(x, r))\)
  - \(x_i \models g\)
  - \(\exists r \in \mathcal{D}_R, x_{i+1} = \mathcal{F}(x_i, r)\)

\[\mathcal{D}_i = \{x_i \mid (x_i, i) \text{ is reachable from an initial state}\}\]
- \(\mathcal{D}_i\) is the distribution of program variables at iteration \(i\)
Example execution

One possible execution:

\[((3,3,0)^T,0)\]
Example execution

One possible execution:

\[((3,3,0)^T,0) \xrightarrow{\tau_1} ((4,5,1)^T,1)\]
Example execution

One possible execution:

\[
((3,3,0)^T,0) \xrightarrow{\tau_1} ((4,5,1)^T,1) \xrightarrow{\tau_1} ((6,7,2)^T,2)
\]
Example execution

One possible execution:

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

One possible execution:

\[(3, 3, 0)^T, 0 \xrightarrow{\tau_1} (4, 5, 1)^T, 1 \xrightarrow{\tau_1} \]
\[(6, 7, 2)^T, 2 \xrightarrow{\tau_2} (6, 7, 2)^T, 3 \xrightarrow{\tau_2} \ldots \]
Pre-Expectations

**Assume we are currently in state** \((x,n)\)
Pre-Expectations

- Assume we are currently in state \((x, n)\)

- What is the expected value of \(e(x')\) evaluated over all successor states \((x', n + 1)\)
  - With respect to a single transition?
  - With respect to all transitions?
Pre-Expectations

- Assume we are currently in state \((x, n)\)

- What is the expected value of \(e(x')\) evaluated over all successor states \((x', n + 1)\)
  - With respect to a single transition?
  - With respect to all transitions?

\[\Rightarrow \text{Pre-expectation of } e(x')\]
Pre-Expectation

Definition (Pre-expectation for fixed PWL transitions)

For a PWL transition $\tau$ the pre-expectation operator can be written as:

$$\text{pre} \mathbb{E}_\tau(e(x')) = \sum_{j=1}^{k} p_j \mathbb{E}_R(\text{pre}(e(x'), f_j) \mid x)$$
Definition (Pre-expectation for fixed PWL transitions)

For a PWL transition $\tau$ the pre-expectation operator can be written as:

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

where $\text{pre}(e(x'), f_j)$ denotes the expression obtained by applying $f_j$ to all variables of $x$ occurring in $e(x)$. 

$\mathbb{E}_R(r)$ denotes the expectation of $r$ over $\mathcal{D}_R$. 

Pre-Expectation
Example

\[ e(x') = 1 + 2x' - 3y' \]
Example

\[ e(x') = 1 + 2x' - 3y' \]

\( \tau_1 : \langle g_1, F_1 \rangle \) with:

- \( g_1 : x + y \leq 10 \)

- \( F_1(x, r) = \begin{cases} 
  f_1 : \begin{pmatrix} x \\ y \\ c \end{pmatrix} + \begin{pmatrix} r_1 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 2 \\ 1 \end{pmatrix}, & p_1 = \frac{3}{4} \\
  f_2 : \begin{pmatrix} x \\ y \\ c \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, & p_2 = \frac{1}{4} 
\end{cases} \)
Example (continued)

\[ \text{pre} E_{T_1}(1 + 2x' - 3y') = \]
Example (continued)

\[
\text{pre} \mathbb{E}_{\mathcal{T}_1}(1 + 2x' - 3y') = \sum_{j=1}^{2} p_j \cdot \mathbb{E}_{\mathcal{D}_R}(\text{pre}(1 + 2x' - 3y', f_j) \mid x)
\]
Example (continued)

\[
\text{pre} \mathbb{E}_{T_1}(1 + 2x' - 3y') = \sum_{j=1}^{2} p_j \cdot \mathbb{E}_{D_R}(\text{pre}(1 + 2x' - 3y', f_j) \mid x) \\
= \frac{3}{4} \cdot \mathbb{E}_{D_R}(1 + 2 \cdot (x + r_1) - 3 \cdot (y + 2)) \\
+ \frac{1}{4} \cdot \mathbb{E}_{D_R}(1 + 2x - 3y)
\]
Example (continued)

\[
\text{preE}_{T_1}(1 + 2x' - 3y') = \sum_{j=1}^{2} p_j \cdot \mathbb{E}_{D_R}(\text{pre}(1 + 2x' - 3y', f_j) \mid x) = \\
\quad = \frac{3}{4} \cdot \mathbb{E}_{D_R}(1 + 2 \cdot (x + r_1) - 3 \cdot (y + 2)) + \frac{1}{4} \cdot \mathbb{E}_{D_R}(1 + 2x - 3y) \\
\quad = \frac{7}{2} + 2x - 3y + \frac{3}{2} \cdot \mathbb{E}_R(r_1)
\]
Example (continued)

\[
\text{pre} \mathbb{E}_{T_1}(1 + 2x' - 3y') = \sum_{j=1}^{2} p_j \cdot \mathbb{E}_{D_R}(\text{pre}(1 + 2x' - 3y', f_j) \mid x)
\]

\[
= \frac{3}{4} \cdot \mathbb{E}_{D_R}(1 + 2 \cdot (x + r_1) - 3 \cdot (y + 2))
\]

\[
+ \frac{1}{4} \cdot \mathbb{E}_{D_R}(1 + 2x - 3y)
\]

\[
= -\frac{7}{2} + 2x - 3y + \frac{3}{2} \cdot \mathbb{E}_R(r_1)
\]

\[
r_1 = U[0,2] \Rightarrow \mathbb{E}_R(r_1) = 1
\]
Example (continued)

\[
\text{pre}\mathbb{E}_{\tau_1}(1 + 2x' - 3y') = \sum_{j=1}^{2} p_j \cdot \mathbb{E}_{D_R}(\text{pre}(1 + 2x' - 3y', f_j) \mid x) \\
= \frac{3}{4} \cdot \mathbb{E}_{D_R}(1 + 2 \cdot (x + r_1) - 3 \cdot (y + 2)) \\
+ \frac{1}{4} \cdot \mathbb{E}_{D_R}(1 + 2x - 3y) \\
= -\frac{7}{2} + 2x - 3y + \frac{3}{2} \cdot \mathbb{E}_R(r_1) \\
\]

\[
r_1 = U[0,2] \Rightarrow \mathbb{E}_R(r_1) = 1 \\
\text{pre}\mathbb{E}_{\tau_1}(1 + 2x - 3y) = -2 + 2x - 3y
\]
Pre-expectation (continued)

**Definition (Pre-expectation over all transitions)**

The expected value of $e$ over the post-state distribution starting from state $(x_n, n)$ is the value of the pre-expectation $\text{pre}E(e')$ evaluated over the current state $(x_n, n)$:

$$E_{D_n}(e) = \text{pre}E(e') = \sum_{\tau_i \in \mathcal{T}} \mathbb{1}_{g_{\tau_i}}(x_n) \cdot \text{pre}E_{\tau_i}(e')$$
Definition (Expectation invariants)

An expression $e$ over the program variables $X$ is called an expectation invariant ($EI$) if and only if $E_{D_i}(e) \geq 0$ for all $i \geq 0$. 

Example

We show that $e = 2y + x$ is an expectation invariant.

1. $E_{D_0}(2y + x) = 2 \cdot E_{D_0}(y) + E_{D_0}(x)$
2. $E_{D_i}(2y + x) = 2 \cdot E_{D_i}(y) + E_{D_i}(x)$

As $E_{D_i}(y)$ is always larger than $E_{D_i}(x)$, $e$ is an expectation invariant of $P$. 

Florian
Expectation Invariants

Definition (Expectation invariants)
An expression $e$ over the program variables $X$ is called an expectation invariant ($\text{EI}$) if and only if $\mathbb{E}_{D_i}(e) \geq 0$ for all $i \geq 0$.

Example
We show that $e(x) = 2y - x$ is an expectation invariant.
Definition (Expectation invariants)

An expression $e$ over the program variables $X$ is called an expectation invariant ($EI$) if and only if $\mathbb{E}_{D_i}(e) \geq 0$ for all $i \geq 0$.

Example

We show that $e(x) = 2y - x$ is an expectation invariant.

$\mathbb{E}_{D_0}(2y - x) = 2 \cdot \mathbb{E}_{D_0}(y) - \mathbb{E}_{D_0}(x) = 3 \geq 0$
Definition (Expectation invariants)

An expression \( e \) over the program variables \( X \) is called an expectation invariant (\( EI \)) if and only if \( \mathbb{E}_{D_i}(e) \geq 0 \) for all \( i \geq 0 \).

Example

We show that \( e(x) = 2y - x \) is an expectation invariant.

1. \( \mathbb{E}_{D_0}(2y - x) = 2 \cdot \mathbb{E}_{D_0}(y) - \mathbb{E}_{D_0}(x) = 3 \geq 0 \)

2. \( \mathbb{E}_{D_i}(2y - x) = 2 \cdot \mathbb{E}_{D_i}(y) - \mathbb{E}_{D_i}(x) \geq 0 \) for all \( i \geq 0 \) as \( \mathbb{E}_{D_i}(y) \) is always larger than \( \mathbb{E}_{D_i}(x) \)
Expectation Invariants

Definition (Expectation invariants)

An expression $e$ over the program variables $X$ is called an expectation invariant ($EI$) if and only if $E_{D_i}(e) \geq 0$ for all $i \geq 0$.

Example

We show that $e(x) = 2y - x$ is an expectation invariant.

1. $E_{D_0}(2y - x) = 2 \cdot E_{D_0}(y) - E_{D_0}(x) = 3 \geq 0$
2. $E_{D_i}(2y - x) = 2 \cdot E_{D_i}(y) - E_{D_i}(x) \geq 0$ for all $i \geq 0$ as $E_{D_i}(y)$ is always larger than $E_{D_i}(x)$

$\Rightarrow e$ is an expectation invariant of $P$
Inductive Expectation Invariants

**Definition (Inductive expectation invariants)**

Let $E = \{e_1, \ldots, e_k\}$ be a set of expressions. The set $E$ forms an **inductive expectation invariant** iff for each $e_j$, $j \in [1,k]$ the following holds:

1. $\mathbb{E}_{\mathcal{D}_0}(e_j) \geq 0$
2. $\text{pre}\mathbb{E}(e_j) = \lambda_0 + \sum_{i=1}^{k} \lambda_i e_i$, $\lambda_i, \geq 0$
Inductive Expectation Invariants

Definition (Inductive expectation invariants)

Let $E = \{e_1, \ldots, e_k\}$ be a set of expressions. The set $E$ forms an \textit{inductive expectation invariant} iff for each $e_j$, $j \in [1, k]$ the following holds:

1. $\mathbb{E}_{\mathcal{D}_0}(e_j) \geq 0$

2. $\text{pre}\mathbb{E}(e_j) = \lambda_0 + \sum_{i=1}^{k} \lambda_i e_i$, $\lambda_i \geq 0$

Theorem

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

Definition (Cones)

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

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

**Definition (Cones)**

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

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

**Theorem**

*If $E$ is an inductive expectation invariant, then $e \in \text{Cone}(E)$ is an expectation invariant.*
Example

\[ E = \{ e_1 : y - x, \ e_2 : 2y + c \} \]

- Without proof \( e_1, e_2 \) are ELs
Example

\[ E = \{ e_1 : y - x, \; e_2 : 2y + c \} \]

- Without proof \( e_1, \; e_2 \) are EIs

Consider \( e = 4y - 2x + c = 2 \cdot e_1 + e_2 \)
Example

\[ E = \{ e_1 : y - x, \ e_2 : 2y + c \} \]
- Without proof, \( e_1, e_2 \) are EIs

Consider \( e = 4y - 2x + c = 2 \cdot e_1 + e_2 \)

\[ \Rightarrow e \in \text{Cone}(E) \]
Example

\[ E = \{ e_1 : y - x, \, e_2 : 2y + c \} \]

- Without proof \( e_1, \, e_2 \) are ELs

Consider \( e = 4y - 2x + c = 2 \cdot e_1 + e_2 \)

\[ \Rightarrow e \in \text{Cone}(E) \]

\[ \Rightarrow e \text{ is an EL} \]
Pre-Expectation of cones

**Definition (Pre-expectation over a single transitions)**

Let \( E = \{e_1, \ldots, e_m\} \) be a set of expressions, and let \( \tau : \langle g, F \rangle \) be a transition. The **pre-expectation** of a cone \( I : \text{Cone}(E) \) with respect to \( \tau \) is defined as:

\[
\text{pre}E_{\tau}(I) = \{(e, \lambda) \in \mathbb{A}(x) \times \mathbb{R}^m | \lambda \geq 0 \land \exists \mu \geq 0(\text{pre}E_{\tau}(e))
\equiv \sum_{j=1}^{m} \lambda_j e_j + \mu)\}
\]
Let $I$ be a finitely generated cone of expressions. The pre-expectation over all transitions in $\mathcal{T} = \{\tau_1, \ldots, \tau_k\}$ can be computed as:

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

Algorithm

Create the initial cone

\[ I_0 = \text{Cone}(\{1, x_1 \in \mathbb{E}\mathbb{D}_0(x_1), \ldots, x_n \in \mathbb{E}\mathbb{D}_0(x_n), \ldots\}) \]

In each iteration

Compute pre\[ \mathbb{E}(I_n) \]

Compute \[ I_{n+1} = \text{pre}\mathbb{E}(I_n) \] \( \setminus I_0 \)

Repeat until \[ I_\leftrightarrow = I_{n+1} = I_n \]

Might not converge

Resulting cone \[ I_\leftrightarrow \] contains only \( \mathbb{E}\mathbb{I}s \)
Fixed points for Expectation Invariants

**Algorithm**

- Create the initial cone $I_0$

$$I_0 : \text{Cone}(\{1, x_1 - \mathbb{E}_{D_0}(x_1), \mathbb{E}_{D_0}(x_1) - x_1, \ldots, x_n - \mathbb{E}_{D_0}(x_n), \mathbb{E}_{D_0}(x_n) - x_n\})$$
Algorithm

- Create the initial cone $I_0$
  \[
  I_0 : \text{Cone} \left( \{ 1, x_1 - \mathbb{E}_{D_0}(x_1), \mathbb{E}_{D_0}(x_1) - x_1, \ldots, x_n - \mathbb{E}_{D_0}(x_n), \mathbb{E}_{D_0}(x_n) - x_n \} \right)
  \]

- In each iteration
  - Compute $\text{pre}\mathbb{E}(I_n)$
  - Compute $I_{n+1} = \text{pre}\mathbb{E}(I_n) \cap I_0$
Algorithm

- Create the initial cone $I_0$
  
  \[
  I_0 : \text{Cone}(\{1, x_1 - \mathbb{E}_{D_0}(x_1), \mathbb{E}_{D_0}(x_1) - x_1, \ldots, x_n - \mathbb{E}_{D_0}(x_n), \mathbb{E}_{D_0}(x_n) - x_n\})
  \]

- In each iteration
  - Compute $\text{pre}_{\mathbb{E}}(I_n)$
  - Compute $I_{n+1} = \text{pre}_{\mathbb{E}}(I_n) \cap I_0$

- Repeat until $I^* = I_{n+1} = I_n$
  - Might not converge
Fixed points for Expectation Invariants

Algorithm

- Create the initial cone $I_0$

  $$I_0 : \text{Cone}(\{1, x_1 - \mathbb{E}_{D_0}(x_1), \mathbb{E}_{D_0}(x_1) - x_1, \ldots, x_n - \mathbb{E}_{D_0}(x_n), \mathbb{E}_{D_0}(x_n) - x_n\})$$

- In each iteration
  - Compute $\text{preE}(I_n)$
  - Compute $I_{n+1} = \text{preE}(I_n) \cap I_0$

- Repeat until $I^* = I_{n+1} = I_n$
  - Might not converge

$\Rightarrow$ Resulting cone $I^*$ contains only EIs
Example

$$I_0 = \text{Cone}\{1, x - \mathbb{E}_{D_0}(x), \mathbb{E}_{D_0}(x) - x, \mathbb{E}_{D_0}(y) - y, y - \mathbb{E}_{D_0}(y),$$
$$\mathbb{E}_{D_0}(c) - c, c - \mathbb{E}_{D_0}(c)\}$$
Example

\[ I_0 = \text{Cone} \left( \{ 1, x - \mathbb{E}_{D_0}(x), \mathbb{E}_{D_0}(x) - x, \mathbb{E}_{D_0}(y) - y, y - \mathbb{E}_{D_0}(y), \mathbb{E}_{D_0}(c) - c, c - \mathbb{E}_{D_0}(c) \} \right) \]
\[ = \text{Cone} \left( \{ 1, x + 1, -1 - x, y - 1, 1 - y, c, -c \} \right) \]
Example

\[
I_0 = \text{Cone}(\{1, x - \mathbb{E}_D(x), \mathbb{E}_D(x) - x, \mathbb{E}_D(y) - y, y - \mathbb{E}_D(y), \\
\mathbb{E}_D(c) - c, c - \mathbb{E}_D(c)\})
\]
\[
= \text{Cone}(\{1, x + 1, -1 - x, y - 1, 1 - y, c, -c\})
\]
\[
\text{pre}_{\mathbb{E}_{\tau_1}}(I_0) = \{(1, (1,0,0,0,0,0)^T), (x + 1, (0.75,1,0,0,0,0)^T), \\
(y - 1, (2.5,0,1,0,0,0)^T), (1 - y, (1.5,0,0,1,0,0)^T), \\
(c,(1,0,0,0,1,0)^T), \ldots\}
\]
Example

\[ I_0 = \text{Cone}(\{1, x - \mathbb{E}_{\mathcal{D}_0}(x), \mathbb{E}_{\mathcal{D}_0}(x) - x, \mathbb{E}_{\mathcal{D}_0}(y) - y, y - \mathbb{E}_{\mathcal{D}_0}(y), \mathbb{E}_{\mathcal{D}_0}(c) - c, c - \mathbb{E}_{\mathcal{D}_0}(c)\}) \]

\[ \quad = \text{Cone}(\{1, x + 1, -1 - x, y - 1, 1 - y, c, -c\}) \]

\[ \text{pre}_{\mathbb{E}_{\tau_1}}(I_0) = \{(1, (1,0,0,0,0)^T), (x + 1, (0.75,1,0,0,0)^T), (y - 1, (2.5,0,1,0,0)^T), (1 - y, (1.5,0,0,1,0,0)^T), (c,(1,0,0,0,1,0)^T), \ldots\} \]

\[ \text{pre}_{\mathbb{E}_{\tau_2}}(I_0) = \{(e, \lambda) \mid \forall e \in I_0, \lambda \text{ corresponding to } e\} \]
Example

\[ l_0 = \text{Cone}(\{1, x - E_{\mathcal{D}_0}(x), E_{\mathcal{D}_0}(x) - x, E_{\mathcal{D}_0}(y) - y, y - E_{\mathcal{D}_0}(y), \]
\[ E_{\mathcal{D}_0}(c) - c, c - E_{\mathcal{D}_0}(c)\}) \]
\[ = \text{Cone}(\{1, x + 1, -1 - x, y - 1, 1 - y, c, -c\}) \]

\[ \text{pre}_{E_{\tau_1}}(l_0) = \{(1, (1,0,0,0,0)^T), (x + 1, (0.75,1,0,0,0)^T), \]
\[ (y - 1, (2.5,0,1,0,0)^T), (1 - y, (1.5,0,0,1,0,0)^T), \]
\[ (c, (1,0,0,0,1,0)^T), \ldots\} \]

\[ \text{pre}_{E_{\tau_2}}(l_0) = \{(e, \lambda) \mid \forall e \in l_0, \lambda \text{ corresponding to } e\} \]

\[ \text{pre}_E(l_0) = \{1, x + 1, y - 1, 1 - y, c\} \]
Example

\[
I_0 = \text{Cone}\{1, x - \mathbb{E}_{D_0}(x), \mathbb{E}_{D_0}(x) - x, \mathbb{E}_{D_0}(y) - y, y - \mathbb{E}_{D_0}(y), \\
\mathbb{E}_{D_0}(c) - c, c - \mathbb{E}_{D_0}(c)\}\)
\[
= \text{Cone}\{1, x + 1, -1 - x, y - 1, 1 - y, c, -c\}
\]
\[
\text{pre}_{\mathbb{E}\tau_1}(I_0) = \{ (1, (1,0,0,0,0)^T), (x + 1, (0.75,1,0,0,0)^T), \\
(1 - y, (1.5,0,0,1,0,0)^T),(c,(1,0,0,0,1,0)^T), \ldots \}
\]
\[
\text{pre}_{\mathbb{E}\tau_2}(I_0) = \{ (e, \lambda) | \forall e \in I_0, \lambda \text{ corresponding to } e \}
\]
\[
\text{pre}_{\mathbb{E}}(I_0) = \{1, x + 1, y - 1, 1 - y, c\}
\]
\[
l_1 = I_0 \cap \text{pre}_{\mathbb{E}}(I_0) = \{1, x + 1, y - 1, 1 - y, c\}
\]
Example

\[ l_0 = \text{Cone}\{1, x - \mathbb{E}_{D_0}(x), \mathbb{E}_{D_0}(x) - x, \mathbb{E}_{D_0}(y) - y, y - \mathbb{E}_{D_0}(y), \mathbb{E}_{D_0}(c) - c, c - \mathbb{E}_{D_0}(c)\} \]
\[ = \text{Cone}\{1, x + 1, -1 - x, y - 1, 1 - y, c, -c\} \]
\[ \text{pre}\mathbb{E}_{\tau_1}(l_0) = \{(1, (1,0,0,0,0,0)^T), (x + 1, (0.75,1,0,0,0,0)^T), (y - 1, (2.5,0,1,0,0,0)^T), (1 - y, (1.5,0,0,1,0,0)^T), (c,(1,0,0,0,1,0)^T), \ldots\} \]
\[ \text{pre}\mathbb{E}_{\tau_2}(l_0) = \{(e, \lambda) \mid \forall e \in l_0, \lambda \text{ corresponding to } e\} \]
\[ \text{pre}\mathbb{E}(l_0) = \{1, x + 1, y - 1, 1 - y, c\} \]
\[ l_1 = l_0 \cap \text{pre}\mathbb{E}(l_0) = \{1, x + 1, y - 1, 1 - y, c\} \]
\[ l^* = l_1 = l_1 \cap \text{pre}\mathbb{E}(l_0) \]
Standard dual widening

Definition (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 $I_1 \widehat{\triangledown} I_2$ is defined as $I = \text{Cone}(g_i \mid g_i \in I_2)$. Cone $I$ is the cone generated by generators of $I_1$ that are also in $I_2$. 
Standard dual widening

**Definition (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 $I_1 \nabla I_2$ is defined as $I = \text{Cone}(g_i \mid g_i \in I_2)$.

Cone $I$ is the cone generated by generators of $I_1$ that are also in $I_2$.

- If $\nabla$ is applied to two successive cones in the algorithm the convergence is ensured.
Example

Assume $\mathcal{P}$ is a probabilistic program s.t. $\pm (x - 1) \in I_0$ and the pre-expectations for these expressions alternate in each iteration.
Example

Assume $\mathcal{P}$ is a probabilistic program s.t. $\pm(x - 1) \in I_0$ and the pre-expectations for these expressions alternate in each iteration.

\[
\begin{align*}
I_1 &= \text{Cone}\{1, x - 1\} = I_3 = \cdots \\
I_2 &= \text{Cone}\{1, -x + 1\} = I_4 = \cdots \\
I_1 \tilde{\nabla} I_2 &= \text{Cone}\{1\} = I^* 
\end{align*}
\]
Example

Assume $\mathcal{P}$ is a probabilistic program s.t. $\pm(x - 1) \in l_0$ and the pre-expectations for these expressions alternate in each iteration.

\[ l_1 = \text{Cone}\{1, x - 1\} = l_3 = \cdots \]
\[ l_2 = \text{Cone}\{1, -x + 1\} = l_4 = \cdots \]
\[ l_1 \tilde{\nabla} l_2 = \text{Cone}\{1\} = l^* \]

$\Rightarrow$ The problem of alternation is solved
### Experimental results from [2]

| $|X|$ | $|\mathcal{T}|$ | #: Number of needed iterations | $\nabla$: Dual widening used or not | Time |
|------|------|-----------------|-----------------|------|
| 3    | 2    | 2               | No              | $\leq \varepsilon$ |
| 3    | 2    | 2               | No              | 0.10 |
| 9    | 2    | 2               | No              | 0.18 |
| 4    | 4    | 7               | Yes             | $\leq \varepsilon$ |
| 3    | 2    | 2               | No              | $\leq \varepsilon$ |
| 3    | 2    | 2               | No              | $\leq \varepsilon$ |
| 2    | 5    | 2               | No              | $\leq \varepsilon$ |
| 6    | 2    | 2               | No              | $\leq \varepsilon$ |
| 6    | 2    | 2               | No              | $\leq \varepsilon$ |
| 2    | 2    | 7               | Yes             | $\leq \varepsilon$ |

$|X|$: Number of program variables

#: Number of needed iterations

Time: Runtime in seconds

$\mathcal{T}$: Number of transitions

$\nabla$: Dual widening used or not

$\varepsilon = 0.05s$
Comparrison with PRINSYS

- Out of 26 tests only 3 IEI could be found by PRINSYS
- 26 IEI could be found with this approach
- Not checked whether PRINSYS finds IEI, that this approach does not find
Conclusion

+ Expectation invariants can be found fast
+ Mostly without usage of dual widening

- Unknown time complexity
- Vague descriptions in the paper
- Implementation is not sufficiently tested
Thanks for your attention!

If you have questions, feel free to ask.
Sources


Dexter Kozen.
Semantics of probabilistic programs.  

Piotr Mardziel, Stephen Magill, Michael Hicks, and Mudhakar Srivatsa.
Dynamic enforcement of knowledge-based security policies.  

Rajeev Motwani and Prabhakar Raghavan.  
*Randomized Algorithms*.  
Dealing with finite loops

- Need to guarantee that exactly one transition can be taken in every iteration
- No problem for infinite loops
- Finite loops need to be modified
  1. Create an infinite loop
  2. Create an if-statement inside
  3. If the original loop-guard is valid execute original loop-body
  4. Else preserve all program variables

⇒ New transition that can be taken after the original loop would have been exited
Example

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