Reasoning about Expected Runtimes of Probabilistic Programs

(and Quantitative Separation Logic, time permitting)

Benjamin Lucien Kaminski

Dagstuhl Seminar 19371: Deduction Beyond Satisfiability

September 11, 2019, Schloss Dagstuhl, Germany
Probabilistic Program Applications
Probabilistic Program Applications

symmetry breaking
Probabilistic Program Applications

- symmetry breaking
- speed–up
Probabilistic Program Applications

- symmetry breaking
- speed–up
- cryptography and security
Probabilistic Program Applications

symmetry breaking

speed–up

cryptography and security

machine learning & AI
Probabilistic Programs

"Ordinary" programs with the additional ability to flip coins

\[
\begin{align*}
\{ & \text{skip} \\
\{ & x := x + 2
\end{align*}
\]

Control flow depends on outcome of coin flips

What does a probabilistic program \( C \) do?

Run program \( C \) on initial state \( \sigma \)

Obtain \((sub-)distribution \( J_C K \sigma \) over final states

\[
\begin{align*}
x & : 5 \\
| & \quad \text{\lvert}\quad | \\
| & \quad \text{\lvert}\quad | \\
x & : 7 \quad \text{\lvert}\quad | \\
x & := x + 2
\end{align*}
\]
Probabilistic Programs

“Ordinary” programs with the additional ability to flip coins

\[
\{\text{skip}\} \left[ \frac{1}{2} \right] \{x := x + 2\}
\]
Probabilistic Programs

- “Ordinary” programs with the additional ability to flip coins

\[
\{\text{skip}\} \left[\frac{1}{2}\right] \{x := x + 2\}
\]

- Control flow depends on outcome of coin flips
Probabilistic Programs

- “Ordinary” programs with the additional ability to flip coins

\[
\{\text{skip}\} \frac{1}{2} \{x := x + 2\}
\]

- Control flow depends on outcome of coin flips
- What does a probabilistic program \( C \) do?
Probabilistic Programs

- “Ordinary” programs with the additional ability to flip coins

\[
\{\text{skip}\} \frac{1}{2} \{x := x + 2\}
\]

- Control flow depends on outcome of coin flips
- What does a probabilistic program \( C \) do?
  - Run program \( C \) on initial state \( \sigma \)
Probabilistic Programs

- “Ordinary” programs with the additional ability to flip coins

\[
\{\text{skip}\} \left[ \frac{1}{2} \right] \{x := x + 2\}
\]

- Control flow depends on outcome of coin flips

- What does a probabilistic program \( C \) do?
  - Run program \( C \) on initial state \( \sigma \)
  - Obtain distribution \([C]_\sigma\) over final states
Probabilistic Programs

- “Ordinary” programs with the additional ability to flip coins

\[
\{\text{skip}\} \left[ \frac{1}{2} \right] \{x := x + 2\}
\]

- Control flow depends on outcome of coin flips

- What does a probabilistic program $C$ do?
  - Run program $C$ on initial state $\sigma$
  - Obtain distribution $[C]_{\sigma}$ over final states

![Diagram of probabilistic program]

- $x: 5$
  - $\frac{1}{2} \mid x: 5$
  - $\frac{1}{2} \mid x: 7$
- $x := x + 2$
Probabilistic Programs

- “Ordinary” programs with the additional ability to flip coins

\[
\{\text{skip}\} \left[ \frac{1}{2} \right] \{x := x + 2\}
\]

- Control flow depends on outcome of coin flips
- What does a probabilistic program \( C \) do?
  - Run program \( C \) on initial state \( \sigma \)
  - Obtain (sub-)distribution \([C]_{\sigma}\) over final states

\begin{align*}
& \text{skip} & & x := x + 2 \\
\frac{1}{2} & x : 5 & \frac{1}{2} & x : 7
\end{align*}
Expected Runtime Phenomena
Expected Runtime Phenomena

- ERT of $C$ can be finite even if $C$ admits infinite computations
Expected Runtime Phenomena

- ERT of $C$ can be **finite** even if $C$ admits **infinite computations**
Expected Runtime Phenomena

- ERT of $C$ can be finite even if $C$ admits infinite computations

$$x := 1; \text{ while } (\frac{1}{2}) \{ x := 2 \cdot x \}$$
Expected Runtime Phenomena

- ERT of $C$ can be finite even if $C$ admits infinite computations

$x := 1; \textbf{while} \left(\frac{1}{2}\right) \{x := 2 \cdot x\}$
Expected Runtime Phenomena

- ERT of $C$ can be finite even if $C$ admits infinite computations
- Positive almost–sure termination:

\[
x := 1; \text{ while } (\frac{1}{2}) \{ x := 2 \cdot x \}
\]
Expected Runtime Phenomena

- ERT of $C$ can be finite even if $C$ admits infinite computations
- Positive almost–sure termination:
  - ERT of $C$ is finite

---

$x := 1; \text{while } (\frac{1}{2}) \{ x := 2 \cdot x \}$
Expected Runtime Phenomena

- ERT of $C$ can be finite even if $C$ admits infinite computations
- Positive almost–sure termination:
  - ERT of $C$ is finite

---

$x := 1; \text{ while } (\frac{1}{2}) \{x := 2 \cdot x\}$

while $(x > 0) \{x := x - 1\}$
Expected Runtime Phenomena

- ERT of $C$ can be finite even if $C$ admits infinite computations
- Positive almost–sure termination:
  - ERT of $C$ is finite
  - Positive almost–sure termination not closed under sequential composition

---

$x := 1; \text{while } (1/2) \{ x := 2 \cdot x \};$
while $(x > 0) \{ x := x - 1 \}$
Expected Runtime Phenomena

- ERT of $C$ can be finite even if $C$ admits infinite computations
- Positive almost–sure termination:
  - ERT of $C$ is finite
  - Positive almost–sure termination not closed under sequential composition
  - Reasoning about positive almost–sure termination is computationally very difficult:

\[
\begin{align*}
x &:= 1; \text{ while } (\frac{1}{2}) \{ x := 2 \cdot x \}; \\
\text{while } (x > 0) \{ x := x - 1 \}
\end{align*}
\]
Expected Runtime Phenomena

- ERT of $C$ can be finite even if $C$ admits infinite computations
- **Positive almost–sure termination:**
  - ERT of $C$ is finite
  - Positive almost–sure termination not closed under sequential composition
  - Reasoning about positive almost–sure termination is computationally very difficult:
    
    Strictly more difficult than termination of non–probabilistic programs \[\text{[MFCS'15]}\]

---

\[
x := 1; \text{ while } \left( \frac{1}{2} \right) \{ x := 2 \cdot x \}; \\
\text{ while } (x > 0) \{ x := x - 1 \}
\]

\[\text{i.e. with probability 1}\]
Expected Runtime Phenomena

- ERT of $C$ can be finite even if $C$ admits infinite computations
- Positive almost–sure termination:
  - ERT of $C$ is finite
  - Positive almost–sure termination not closed under sequential composition
  - Reasoning about positive almost–sure termination is computationally very difficult:
    
    Strictly more difficult than termination of non–probabilistic programs [MFCS’15]

- ERT of $C$ can be infinite, even if $C$ terminates almost–surely\(^1\)

\[
x := 1; \text{ while } (1/2) \{ x := 2 \cdot x \}; \text{ while } (x > 0) \{ x := x - 1 \}
\]

\(^1\)i.e. with probability 1
Weakest Precondition Reasoning for Expected Runtimes
Domain for Expected Runtimes
Domain for Expected Runtimes

- ERT of $C$ on input $\sigma$:

\[ \sum_{i=1}^{\infty} i \cdot \Pr \left( \text{"$C$ terminates after $i$ steps on input $\sigma$"} \right) \]
Domain for Expected Runtimes

- ERT of $C$ on input $\sigma$:
  \[\sum_{i=1}^\infty i \cdot \Pr("C \text{ terminates after } i \text{ steps on input } \sigma")\]

- Set of runtimes $T = \{ t \mid t: \text{States} \rightarrow \mathbb{R}_{\geq 0}\}$
Domain for Expected Runtimes

- ERT of $C$ on input $\sigma$:
  \[ \sum_{i=1}^{\infty} i \cdot \Pr \left( \text{"$C$ terminates after $i$ steps on input $\sigma$"} \right) \]

- Set of runtimes $\mathbb{T} = \{ t \mid t: \text{States} \rightarrow \mathbb{R}_{\geq 0} \}$

- associate runtimes (quantities) to program states
Domain for Expected Runtimes

- ERT of $C$ on input $\sigma$:

$$\sum_{i=1}^{\infty} i \cdot \Pr\left(\text{"$C$ terminates after } i \text{ steps on input } \sigma\"\right)$$

- Set of runtimes $\mathbb{T} = \{ t \mid t: \text{States} \rightarrow \mathbb{R}_{\geq 0}\}$, a complete lattice with partial order

$$r \preceq t \iff \forall \sigma \in \text{States}: r(\sigma) \leq t(\sigma)$$

- associate runtimes (quantities) to program states
Weakest Precondition Reasoning for Expected Runtimes

The ert Transformer [ESOP’16, J.ACM’18]
Weakest Precondition Reasoning for Expected Runtimes

The ert Transformer [ESOP’16, J.ACM’18]

Use a continuation passing style ERT transformer ert[\(C\)] : \(\mathbb{T} \rightarrow \mathbb{T}\).
Weakest Precondition Reasoning for Expected Runtimes

The ert Transformer [ESOP’16, J.ACM’18]

Use a continuation passing style ERT transformer $\text{ert}[C] : \mathbb{T} \rightarrow \mathbb{T}$.

$C$
Weakest Precondition Reasoning for Expected Runtimes

The ert Transformer [ESOP’16, J.ACM’18]

Use a continuation passing style ERT transformer \( ert[C] : \mathbb{T} \rightarrow \mathbb{T} \).

\[
\begin{array}{cc}
C & t \\
\uparrow & \\
\text{time needed} & \\
\text{after executing} & \end{array}
\]

\( C \) is a priori expected time needed to execute \( C \) and afterwards let time \( t \) pass.
Weakest Precondition Reasoning for Expected Runtimes

The ert Transformer [ESOP’16, J.ACM’18]

Use a continuation passing style ERT transformer \( \text{ert}[C] : T \rightarrow T \).

\[
\begin{aligned}
C' & \quad t \\
\text{time needed} & \quad \text{after executing } C'
\end{aligned}
\]
Weakest Precondition Reasoning for Expected Runtimes

The ert Transformer [ESOP’16, J.ACM’18]

Use a continuation passing style ERT transformer $\text{ert}[C] : T \rightarrow T$.

$$\text{ert}[C](t) \quad C \quad t$$

- a priori expected time needed to execute $C$ and afterwards let time $t$ pass
- time needed after executing $C$
Weakest Precondition Reasoning for Expected Runtimes

The ert Transformer [ESOP'16, J.ACM'18]

Use a continuation passing style ERT transformer \( \text{ert}[C] : T \rightarrow T \).

\[
\begin{align*}
\text{ert}[C](t) & \quad \text{a priori expected time needed to execute } C \\
C & \quad \text{and afterwards let time } t \text{ pass} \\
t & \quad \text{time needed after executing } C
\end{align*}
\]

ERT in Terms of ert

\[
\text{ert}[C](\text{0})(\sigma) = \text{"ERT of } C \text{ on input } \sigma\"
\]
Weakest Precondition Reasoning for Expected Runtimes

The ert Transformer

ert of Skipping

\[ ert_{\text{skip}}(t) = 1 + t \]

This represents the time needed after executing `skip` a priori expected time needed to skip and then let time pass.
ert of Skipping

\texttt{ert[skip]}(t) = 1 + t
Weakest Precondition Reasoning for Expected Runtimes

The ert Transformer

ert of Skipping

\[ \text{ert}[\text{skip}](t) = 1 + t \]

skip
Weakest Precondition Reasoning for Expected Runtimes

The ert Transformer

ert of Skipping

\[ \text{ert}[\text{skip}](t) = 1 + t \]

\[ \text{skip} \quad t \]

\text{time needed after executing skip}
Weakest Precondition Reasoning for Expected Runtimes

The ert Transformer

ert of Skipping

\[ ert[\text{skip}](t) = 1 + t \]

- skip
- time needed after executing skip

Benjamin Lucien Kaminski
Reasoning about Expected Runtimes of Probabilistic Programs 11.9.2017
ert of Skipping

\[ \text{ert}[\text{skip}](t) = 1 + t \]

1. \[1 + t\] a priori expected time needed to skip and then let time \(t\) pass
2. \[\text{skip}\] time needed after executing \text{skip}
3. \[t\]
ert of Assignments

\[
J(x) = 1 + t[x/E^2] + 1 + (z + 1)^2
\]

Time needed after executing \(x := z + 1\) is the a priori expected time needed to assign \(x\) to \(z + 1\) and then let time \(x^2\) pass.
ert of Assignments

\[
\text{ert} \left[ x := E \right] (t) = 1 + t \left[ x/E \right]
\]
ert of Assignments

\[
\text{ert} \left[ x := E \right] (t) = 1 + t \left[ x/E \right]
\]

\[
x := z + 1
\]
Weakest Precondition Reasoning for Expected Runtimes

The ert Transformer

ert of Assignments

\[ \text{ert} \left[ x := E \right] (t) = 1 + t \left[ x/E \right] \]

\[ x := z + 1 \]

\[ x^2 \]

Time needed after executing \( x := z + 1 \)
Weakest Precondition Reasoning for Expected Runtimes

The ert Transformer

### ert of Assignments

\[
\text{ert}[x := E](t) = 1 + t[x/E]
\]

- \( x := z + 1 \)
- \( x^2 \)
  - time needed after executing \( x := z + 1 \)
Weakest Precondition Reasoning for Expected Runtimes

The ert Transformer

\[ \text{ert} \left[ x := E \right] (t) = 1 + t \left[ x/E \right] \]

\[ 1 + (z + 1)^2 \]

\[ x := z + 1 \]

\[ x^2 \]

time needed after executing \( x := z + 1 \)
Weakest Precondition Reasoning for Expected Runtimes

The ert Transformer

**ert of Assignments**

\[
  \text{ert} \left[ x := E \right] (t) = 1 + t \left[ x / E \right]
\]

\[
  2 + 2z + z^2 \quad \quad \quad x := z + 1 \quad \quad \quad x^2
\]

Time needed after executing \( x := z + 1 \)
Weakest Precondition Reasoning for Expected Runtimes

The ert Transformer

ert of Assignments

\[
\text{ert} \left[ x := E \right] (t) = 1 + t \left[ x/E \right]
\]

\( 2 + 2z + z^2 \)  \hspace{1cm}  \( x := z + 1 \)  \hspace{1cm}  \( x^2 \)

a priori expected time

needed to assign \( x \) to \( z + 1 \)

and then let time \( x^2 \) pass

time needed

after executing \( x := z + 1 \)
Rules for the ert Transformer

\[ C \quad \text{ert} \left[ C \right] (t) \]
## Rules for the ert Transformer

<table>
<thead>
<tr>
<th>$C$</th>
<th>$\text{ert} [C] (t)$</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>skip</code></td>
<td>$1 + t$</td>
</tr>
<tr>
<td><code>x := E</code></td>
<td>$1 + t[x/E]$</td>
</tr>
</tbody>
</table>

The table above presents the rules for the ert Transformer.

- **skip**: The expected runtime is $1 + t$.
- **$x := E$**: The expected runtime is $1 + t[x/E]$. 

These rules help in reasoning about expected runtimes of probabilistic programs.

---

Weakest Precondition Reasoning for Expected Runtimes

The ert Transformer

Rules for the ert Transformer

- `skip`: $1 + t$
- `x := E`: $1 + t[x/E]$

Benjamin Lucien Kaminski

Reasoning about Expected Runtimes of Probabilistic Programs

11.9.2017
## Rules for the ert Transformer

<table>
<thead>
<tr>
<th>$C$</th>
<th>$\text{ert} \left[ C \right] (t)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>skip</td>
<td>$1 + t$</td>
</tr>
<tr>
<td>$x := E$</td>
<td>$1 + t[x/E]$</td>
</tr>
<tr>
<td>$C_1; C_2$</td>
<td>$\text{ert} \left[ C_1 \right] \left( \text{ert} \left[ C_2 \right] (t) \right)$</td>
</tr>
</tbody>
</table>
Rules for the ert Transformer

<table>
<thead>
<tr>
<th>$C$</th>
<th>$\text{ert} [C] (t)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>skip</td>
<td>$1 + t$</td>
</tr>
<tr>
<td>$x := E$</td>
<td>$1 + t [x/E]$</td>
</tr>
<tr>
<td>$C_1; C_2$</td>
<td>$\text{ert} [C_1] (\text{ert} [C_2] (t))$</td>
</tr>
<tr>
<td>${C_1} [p] {C_2}$</td>
<td>$1 + p \cdot \text{ert} [C_1] (t) + (1 - p) \cdot \text{ert} [C_2] (t)$</td>
</tr>
</tbody>
</table>
## Rules for the ert Transformer

<table>
<thead>
<tr>
<th>$C$</th>
<th>$\text{ert} \left[ C \right] (t)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>skip</td>
<td>$1 + t$</td>
</tr>
<tr>
<td>$x := E$</td>
<td>$1 + t \left[ x/E \right]$</td>
</tr>
<tr>
<td>$C_1; C_2$</td>
<td>$\text{ert} \left[ C_1 \right] \left( \text{ert} \left[ C_2 \right] (t) \right)$</td>
</tr>
<tr>
<td>${ C_1 } \ [p] \ { C_2 }$</td>
<td>$1 + p \cdot \text{ert} \left[ C_1 \right] (t) + (1 - p) \cdot \text{ert} \left[ C_2 \right] (t)$</td>
</tr>
<tr>
<td>if ($\varphi$) ${ C_1 }$ else ${ C_2 }$</td>
<td>$1 + \left[ \varphi \right] \cdot \text{ert} \left[ C_1 \right] (t) + \left[ \neg \varphi \right] \cdot \text{ert} \left[ C_2 \right] (t)$</td>
</tr>
</tbody>
</table>
### Rules for the ert Transformer

<table>
<thead>
<tr>
<th>Rule</th>
<th>ert (<a href="t">C</a>)</th>
</tr>
</thead>
<tbody>
<tr>
<td>skip</td>
<td>(1 + t)</td>
</tr>
<tr>
<td>(x := E)</td>
<td>(1 + t \cdot \frac{x}{E})</td>
</tr>
<tr>
<td>(C_1; C_2)</td>
<td>ert (<a href="t">C_1</a>) \cdot \text{ert} (<a href="t">C_2</a>)</td>
</tr>
<tr>
<td>{(C_1) [p] {(C_2)}</td>
<td>(1 + p \cdot \text{ert} (<a href="t">C_1</a>) + (1 - p) \cdot \text{ert} (<a href="t">C_2</a>)</td>
</tr>
<tr>
<td>if((\varphi)){(C_1)} else {(C_2)}</td>
<td>(1 + [\varphi] \cdot \text{ert} (<a href="t">C_1</a>) + [\neg \varphi] \cdot \text{ert} (<a href="t">C_2</a>)</td>
</tr>
<tr>
<td>while((\varphi)){(C)'}</td>
<td>lfp (X). (1 + [\neg \varphi] \cdot t + [\varphi] \cdot \text{ert} (<a href="X">C'</a>)</td>
</tr>
</tbody>
</table>
The ert Transformer in Action

\[
\{ x := 5 \} \frac{1}{2} \{ x := 2 \} ;
\]

\[
\text{while } (x > 0) \{ \{ x := x - 1 \} \frac{1}{2} \{ \text{skip} \} \} ;
\]

\text{skip}
The ert Transformer in Action

\{x := 5\} \frac{1}{2} \{x := 2\} ;

\textbf{while} (x > 0) \{ \{x := x - 1\} \frac{1}{2} \{\text{skip}\} \} ;

\text{skip}
The ert Transformer in Action

\[
\{ x := 5 \} \frac{1}{2} \{ x := 2 \} ;
\]

\[
\text{while} \ (x > 0) \ \{ \{ x := x - 1 \} \frac{1}{2} \{ \text{skip} \} \} ;
\]

\text{skip}

// 0
Weakest Precondition Reasoning for Expected Runtimes

The ert Transformer

The ert Transformer in Action

\[
\{ x := 5 \} \frac{1}{2} \{ x := 2 \} ;
\]

\[
\text{while (} x > 0 \text{) } \{ \{ x := x - 1 \} \frac{1}{2} \{ \text{skip} \} \};
\]

// 1 + 0

skip

// 0
The ert Transformer in Action

\{x := 5\} \frac{1}{2} \{x := 2\};

while (x > 0) \{ \{x := x - 1\} \frac{1}{2} \{\text{skip}\}\};

// 1

\text{skip}

// 0
The ert Transformer in Action

\{ x := 5 \} \frac{1}{2} \{ x := 2 \} ;

while (x > 0) \{ x := x - 1 \} \frac{1}{2} \{ \text{skip} \} ;

// 1

skip

// 0
The ert Transformer in Action

\[
\{ x := 5 \} \frac{1}{2} \{ x := 2 \};
\]

// 2 + \([x > 0]\) \cdot 6x

while \((x > 0)\) \{ \{ x := x - 1 \} \frac{1}{2} \{ \text{skip} \} \};

// 1

skip

// 0
The ert Transformer in Action

\[
\{ x := 5 \} \left[ \frac{1}{2} \right] \{ x := 2 \};
\]

\[
// \ 2 + [x > 0] \cdot 6x
\]

while \((x > 0)\) \(\{\{x := x - 1\} \left[ \frac{1}{2} \right] \{\text{skip}\}\}\);  

\[
// \ 1
\]

skip

\[
// \ 0
\]
The ert Transformer in Action

\[
\begin{align*}
\text{// } & \quad 1 + \frac{1}{2}(2 + [5 > 0] \cdot 6 \cdot 5 + 2 + [2 > 0] \cdot 6 \cdot 2) \\
\{&x := 5\} [1/2] \{x := 2\} ; \\
\text{// } & \quad 2 + [x > 0] \cdot 6x \\
\text{while } (x > 0) \{\{x := x - 1\} [1/2] \{\text{skip}\} \}; \\
\text{// } & \quad 1 \\
\text{skip} \\
\text{// } & \quad 0
\end{align*}
\]
The ert Transformer in Action

// 24

\{ x := 5 \} \frac{1}{2} \{ x := 2 \};

// 2 + [x > 0] \cdot 6x

while (x > 0) \{ \{ x := x - 1 \} \frac{1}{2} \{ \text{skip} \} \};

// 1

skip

// 0
Induction for Loops
Induction for ert of Loops
Induction for ert of Loops

Definition of ert:

\[
\text{ert}[\text{while } (\varphi) \{ C \}] (t) = \text{lfp } X \cdot \left( 1 + [\neg \varphi] \cdot t + [\varphi] \cdot \text{ert}[\{ C \}] (X) \right) =: \Phi(X)
\]

\[
= \text{lfp } \Phi
\]
Induction for ert of Loops

- **Definition of ert:**

\[
\text{ert} \left[ \text{while} \left( \varphi \right) \{ C \} \right] (t) = \text{lfp} X \cdot 1 + \left[ \neg \varphi \right] \cdot t + \left[ \varphi \right] \cdot \text{ert} \left[ C \right] (X) =: \Phi(X)
\]

- **\( \Phi \) monotonic** implies **lfp exists** [Knaster’28, Tarski’49, Kleene’62, *Folklore*]

- **Problem:** **lfp \( \Phi \) highly non–computable!** [MFCS’15, Acta Informatica’18]
Induction for ert of Loops

Definition of ert:

\[
\text{ert}[\text{while}(\varphi)\{C\}](t) = \text{lfp } X \cdot (1 + [\neg \varphi] \cdot t + [\varphi] \cdot \text{ert}[C](X)) =: \Phi(X)
\]

- \(\Phi\) monotonic implies \(\text{lfp}\) exists \([\text{Knaster}'28, \text{Tarski}'49, \text{Kleene}'62, \text{Folklore}]\)
- Problem: \(\text{lfp } \Phi\) highly non–computable! \([\text{MFCS}'15, \text{Acta Informatica}'18]\)
- Reasoning about \(\text{lfp}\) needed: Induction! \([\text{Park }'69]\)

\[\Phi(I) \preceq I\] implies \(\text{lfp } \Phi \preceq I\)
**Induction Example**

```plaintext
while (x > 0)
    {x := x − 1} [1/2] {skip}

Postruntime: 1 (expected time of executing the loop and then needing 1 time unit)

Characteristic function

Φ(x) = 2 + (x > 0) (1 + 1/2 (x[x−1] + x))

Candidate for upper bound:

I = 2 + (x > 0) 6 x

Induction:

Φ(I) = 2 + (x > 0) (1 + 1/2 (2 + (x > 0) 6 (x − 1) + 2 + (x > 0) 6 x))

= 2 + (x > 0) 6 x

Thus: ert \( \text{while } \ldots \text{K} (1) \preceq 2 + (x > 0) 6 x \preceq 7 x \)
Induction Example

\[\text{while } (x > 0) \]
\[\{ x := x - 1 \} [1/2] \{ \text{skip} \} \]
\[\} \]

Postruntime: 1 (expected time of executing the loop and then needing 1 time unit)
**Induction Example**

\[
\text{while } (x > 0) \\
\{ x := x - 1 \} \left[ \frac{1}{2} \right] \{ \text{skip} \}
\]

Postruntime: 1 (expected time of executing the loop and then needing 1 time unit)

Characteristic function \( \Phi(X) = 2 + [x > 0] \left( 1 + \frac{1}{2} (X \left[ x/x - 1 \right] + X) \right) \)
Induction Example

\[
\text{while } (x > 0) \\
\{ x := x - 1 \} \left[ \frac{1}{2} \right] \{ \text{skip} \}
\]

Postruntime: 1 (expected time of executing the loop and then needing 1 time unit)

Characteristic function \( \Phi(X) = 2 + [x > 0] \left( 1 + \frac{1}{2} (X [x/x - 1] + X) \right) \)

Candidate for upper bound: \( I = 2 + [x > 0] \cdot 6x \)
Induction Example

\[
\text{while } (x > 0) \\
\{ x := x - 1 \} [1/2] \{ \text{skip} \}
\]

Postruntime: 1 \text{ (expected time of executing the loop and then needing 1 time unit)}

Characteristic function \( \Phi(X) = 2 + [x > 0] \left( 1 + \frac{1}{2} \left( X \left[ x/x - 1 \right] + X \right) \right) \)

Candidate for upper bound: \( I = 2 + [x > 0] 6x \)

Induction:
Induction Example

\[
\text{while } (x > 0) \\
\{ x := x - 1 \} \frac{1}{2} \{ \text{skip} \}
\]

Postruntime: 1 (expected time of executing the loop and then needing 1 time unit)

Characteristic function \( \Phi(X) = 2 + [x > 0] \left( 1 + \frac{1}{2} \left( X [x/x - 1] + X \right) \right) \)

Candidate for upper bound: \( I = 2 + [x > 0] 6x \)

Induction: \( \Phi(I) \)
Induction Example

\[
\text{while } (x > 0) \\
\{x := x - 1\} \left\{1/2 \right\} \{\text{skip}\}
\]

Postruntime: 1 (expected time of executing the loop and then needing 1 time unit)

Characteristic function \( \Phi(X) = 2 + [x > 0] \left( 1 + \frac{1}{2} (X [x/x - 1] + X) \right) \)

Candidate for upper bound: \( I = 2 + [x > 0] 6x \)

Induction: \( \Phi(I) = 2 + [x > 0] \left( 1 + \frac{1}{2} \right) \)
Induction Example

\[
\text{while } (x > 0) \\
\{x := x - 1\} \frac{1}{2} \{\text{skip}\}
\]

Postruntime: 1 (expected time of executing the loop and then needing 1 time unit)

Characteristic function \(\Phi(X) = 2 + [x > 0] \left(1 + \frac{1}{2} (X[x/x - 1] + X)\right)\)

Candidate for upper bound: \(I = 2 + [x > 0] \cdot 6x\)

Induction: \(\Phi(I) = 2 + [x > 0] \left(1 + \frac{1}{2} \left(2 + [x - 1 > 0] \cdot 6(x - 1)\right)\right)\)
Induction Example

\[
\text{while } (x > 0) \\
\{ x := x - 1 \} \frac{1}{2} \{ \text{skip} \}
\]

Postruntime: 1 (expected time of executing the loop and then needing 1 time unit)

Characteristic function \( \Phi(X) = 2 + [x > 0] \left( 1 + \frac{1}{2}(X \left[ x/x - 1 \right] + X) \right) \)

Candidate for upper bound: \( I = 2 + [x > 0] 6x \)

Induction: \( \Phi(I) = 2 + [x > 0] \left( 1 + \frac{1}{2}(2 + [x - 1 > 0] 6(x - 1) + 2 + [x > 0] 6x) \right) \)
**Induction Example**

```plaintext
while (x > 0)
    {x := x - 1} [1/2] {skip}

Postruntime: 1 (expected time of executing the loop and then needing 1 time unit)

Characteristic function \( \Phi(X) = 2 + [x > 0] \left( 1 + \frac{1}{2} (X [x/x - 1] + X) \right) \)

Candidate for upper bound: \( I = 2 + [x > 0] 6x \)

Induction: \( \Phi(I) = 2 + [x > 0] \left( 1 + \frac{1}{2} (2 + [x - 1 > 0] 6(x - 1) + 2 + [x > 0] 6x) \right) \)

= \( 2 + [x > 0] 6x \)
Induction Example

```
while (x > 0)
    {x := x - 1} [1/2] {skip}
}
```

Postruntime: 1 (expected time of executing the loop and then needing 1 time unit)

Characteristic function $\Phi(X) = 2 + [x > 0] \left( 1 + \frac{1}{2} (X [x/x - 1] + X) \right)$

Candidate for upper bound: $I = 2 + [x > 0] 6x$

Induction: $\Phi(I) = 2 + [x > 0] \left( 1 + \frac{1}{2} \left( 2 + [x - 1 > 0] 6(x - 1) + 2 + [x > 0] 6x \right) \right)$

$= 2 + [x > 0] 6x = I$
Induction Example

\[\text{while } (x > 0)\]
\[\{ x := x - 1 \} \begin{bmatrix} \frac{1}{2} \end{bmatrix} \{ \text{skip} \} \]
\]

Postruntime: 1 (expected time of executing the loop and then needing 1 time unit)

Characteristic function \( \Phi(X) = 2 + [x > 0] \left( 1 + \frac{1}{2} (X [x/x - 1] + X) \right) \)

Candidate for upper bound: \( I = 2 + [x > 0] 6x \)

Induction: \( \Phi(I) = 2 + [x > 0] \left( 1 + \frac{1}{2} (2 + [x - 1 > 0] 6(x - 1) + 2 + [x > 0] 6x) \right) \)

\[= 2 + [x > 0] 6x = I \leq I \]
Induction Example

```plaintext
while (x > 0)
    {x := x - 1} [1/2] {skip}
```

Postruntime: 1 (expected time of executing the loop and then needing 1 time unit)

Characteristic function \( \Phi(X) = 2 + [x > 0] \left(1 + \frac{1}{2} (X [x/x - 1] + X)\right) \)

Candidate for upper bound: \( I = 2 + [x > 0] 6x \)

Induction: \( \Phi(I) = 2 + [x > 0] \left(1 + \frac{1}{2} (2 + [x - 1 > 0] 6(x - 1) + 2 + [x > 0] 6x)\right) \)

\[ = 2 + [x > 0] 6x = I \leq I \]
Induction Example

\[ \text{while } (x > 0) \]
\[ \{ x := x - 1 \} \{1/2 \} \{ \text{skip} \} \]

Postruntime: 1 (expected time of executing the loop and then needing 1 time unit)

Characteristic function \( \Phi(X) = 2 + [x > 0] \left( 1 + \frac{1}{2} \left( X \frac{x}{x-1} + X \right) \right) \)

Candidate for upper bound: \( I = 2 + [x > 0] 6x \)

Induction: \( \Phi(I) = 2 + [x > 0] \left( 1 + \frac{1}{2} \left( 2 + [x - 1 > 0] 6(x - 1) + 2 + [x > 0] 6x \right) \right) \)
\[ = 2 + [x > 0] 6x = I \preceq I \checkmark \]
Induction Example

\[ \text{while}(x > 0) \]
\[ \{x := x - 1\} \frac{1}{2} \{\text{skip}\} \]

Postruntime: 1 (expected time of executing the loop and then needing 1 time unit)

Characteristic function \( \Phi(X) = 2 + [x > 0] \left( 1 + \frac{1}{2} (X[x/x - 1] + X) \right) \)

Candidate for upper bound: \( I = 2 + [x > 0] 6x \)

Induction:
\[
\Phi(I) = 2 + [x > 0] \left( 1 + \frac{1}{2} (2 + [x - 1 > 0] 6(x - 1) + 2 + [x > 0] 6x) \right) \\
= 2 + [x > 0] 6x = I \preceq I \checkmark
\]

Thus: \( \text{ert}[\text{while}...](1) \preceq 2 + [x > 0] 6x \)
Induction Example

\[
\text{while } (x > 0) \\
\{ x := x - 1 \} \ \frac{1}{2} \ \{ \text{skip} \} \\
\}
\]

Postruntime: 1 (expected time of executing the loop and then needing 1 time unit)

Characteristic function \( \Phi(X) = 2 + [x > 0] \left( 1 + \frac{1}{2} (X [x/x - 1] + X) \right) \)

Candidate for upper bound: \( I = 2 + [x > 0] 6x \)

Induction:
\[
\Phi(I) = 2 + [x > 0] \left( 1 + \frac{1}{2} (2 + [x - 1 > 0] 6(x - 1) + 2 + [x > 0] 6x) \right) \\
= 2 + [x > 0] 6x = I \leq I \checkmark
\]

Thus:
\[
\text{ert [while...]}(1) \leq 2 + [x > 0] 6x \leq 7x
\]
Work building on the ert calculus:
Work building on the ert calculus:

- ert calculus for recursive probabilistic programs [LICS’16]
Work building on the ert calculus:

- ert calculus for recursive probabilistic programs [LICS’16]
- ert calculus has been Isabelle/HOL certified [Hölzl'16]
Work building on the ert calculus:

- ert calculus for recursive probabilistic programs [LICS’16]
- ert calculus has been Isabelle/HOL certified [Hölzl’16]
- ert calculus was used to prove soundness of an automated technique for inferring expected runtimes [Ngo et al.’18]
Weakest Precondition Reasoning for Expected Runtimes

Related and Future Work

Work building on the ert calculus:

- ert calculus for recursive probabilistic programs [LICS’16]

ert calculus has been Isabelle/HOL certified [Hölzl’16]

- ert calculus was used to prove soundness of an automated technique for inferring expected runtimes [Ngo et al.’18]

- A fairly simple (but incomplete) rule for lower bounds [{arXiv’19} {$\varepsilon$} {POPL’20}]
Work building on the ert calculus:

- ert calculus for recursive probabilistic programs [LICS’16]
- ert calculus has been Isabelle/HOL certified [Hölzl’16]
- ert calculus was used to prove soundness of an automated technique for inferring expected runtimes [Ngo et al.’18]
- A fairly simple (but incomplete) rule for lower bounds [{arXiv’19} [ε] {POPL’20}]

Open Problems (for this community?):
Work building on the ert calculus:

- ert calculus for recursive probabilistic programs [LICS’16]
- ert calculus has been Isabelle/HOL certified [Hölzl’16]
- ert calculus was used to prove soundness of an automated technique for inferring expected runtimes [Ngo et al.’18]
- A fairly simple (but incomplete) rule for lower bounds [{arXiv’19} [e] {POPL’20}]

Open Problems (for this community?):

- Quantitative “entailment” checking: Does $\Phi(I) \preceq I$ hold?
Work building on the ert calculus:

- ert calculus for recursive probabilistic programs [LICS’16]
- ert calculus has been Isabelle/HOL certified [Hölzl’16]
- ert calculus was used to prove soundness of an automated technique for inferring expected runtimes [Ngo et al.’18]
- A fairly simple (but incomplete) rule for lower bounds [{arXiv’19} [ε] {POPL’20}]

Open Problems (for this community?):

- Quantitative “entailment” checking: Does $\Phi(I) \preceq I$ hold?
- Synthesis of quantitative invariants: Find $I$, such that $\Phi(I) \preceq I$!
Work building on the ert calculus:

- ert calculus for recursive probabilistic programs [LICS’16]
- ert calculus has been Isabelle/HOL certified [Hölzl’16]
- ert calculus was used to prove soundness of an automated technique for inferring expected runtimes [Ngo et al.’18]
- A fairly simple (but incomplete) rule for lower bounds [{arXiv’19} [ε] {POPL’20}]

Open Problems (for this community?):

- Quantitative “entailment” checking: Does $\Phi(I) \preceq I$ hold?
- Synthesis of quantitative invariants: Find $I$, such that $\Phi(I) \preceq I$!
- Verifying lower bounds: Does $I \preceq \text{lfp } \Phi$ hold?
Quantitative Separation Logic
Quantitative Separating Conjunction [POPL’19]

Classical separating conjunction:

\[(s, h) \models F \star G \iff \exists h_1, h_2: h = h_1 \star h_2 \text{ and } (s, h_1) \models F \text{ and } (s, h_2) \models G\]
Quantitative Separating Conjunction [POPL’19]

Classical separating conjunction:

\[(s, h) \models F \star G \text{ iff } \exists h_1, h_2: \ h = h_1 \star h_2 \text{ and } (s, h_1) \models F \text{ and } (s, h_2) \models G\]

Naive quantitative separating conjunction:

\[(f \star g)(s, h) = \exists h_1, h_2: [h = h_1 \star h_2] \cdot f(s, h_1) \cdot g(s, h_2)\]
Quantitative Separating Conjunction [POPL’19]

Classical separating conjunction:

\[(s, h) \models F \star G \iff \exists h_1, h_2: h = h_1 \star h_2 \text{ and } (s, h_1) \models F \text{ and } (s, h_2) \models G\]

Naive quantitative separating conjunction:

\[(f \star g)(s, h) = \exists h_1, h_2: [h = h_1 \star h_2] \cdot f(s, h_1) \cdot g(s, h_2)\]

Meaningful quantitative separating conjunction:

\[f \star g = \lambda(s, h) \cdot \max_{h_1, h_2} \{ f(s, h_1) \cdot g(s, h_2) \mid h = h_1 \star h_2 \}\]
Quantitative Separating Implication [POPL’19]

Classical separating implication:

\[(s, h) \models F \rightarrow G \iff \forall h' \text{ with } h' \perp h \land (s, h') \models F : (s, h \ast h') \models G\]
Quantitative Separating Implication [POPL’19]

Classical separating implication:

\[(s, h) \models F \rightarrow G \text{ if and only if } \forall h' \text{ with } h' \perp h \land (s, h') \models F : (s, h \ast h') \models G\]

Quantitative separating implication:

\[f \rightarrow g = \lambda(s, h) \cdot \inf_{h'} \left\{ \frac{g(s, h \ast h')}{f(s, h')} \mid h' \perp h \text{ and } f(s, h') > 0 \text{ and } \not\text{not } f(s, h') = \infty = g(s, h \ast h') \right\} \]
Quantitative Separating Implication [POPL’19]

Classical separating implication:

\[(s, h) \models F \rightarrow G \iff \forall h' \text{ with } h' \perp h \land (s, h') \models F: (s, h \ast h') \models G\]

Quantitative separating implication:

\[f \rightarrow g = \lambda(s, h) \cdot \inf_{h'} \left\{ \frac{g(s, h \ast h')}{f(s, h')} \mid h' \perp h \text{ and } f(s, h') > 0 \text{ and } \not\exists f(s, h') = \infty = g(s, h \ast h') \right\}\]

Notice:

\[[F] \rightarrow g = \lambda(s, h) \cdot \inf_{h'} \left\{ g(s, h \ast h') \mid h' \perp h \text{ and } (s, h') \models F \right\}\]
Modus Ponens

Classical modus ponens:

\[ F \land (F \implies G) \implies G \]
Modus Ponens

Classical modus ponens:

\[ F \land (F \implies G) \implies G \]

Modus ponens of classical separation logic:

\[ F \star (F \rightarrow\star G) \implies G \]
Modus Ponens

Classical modus ponens:

$$F \land (F \implies G) \implies G$$

Modus ponens of classical separation logic:

$$F \triangleright (F \triangleright G) \implies G$$

Modus ponens of quantitative separation logic:

$$f \triangleright (f \triangleright g) \preceq g$$
Quantitative Separation Logic

Adjointness of $\rightarrow^\star$ and $\star$

Classical adjointness:

$((F \star G) \Rightarrow J) \iff (F \Rightarrow (G \rightarrow J))$
Adjointness of \(\rightarrow\) and \(\star\)

Classical adjointness:

\[
((F \star G) \Rightarrow J) \iff (F \Rightarrow (G \rightarrow J))
\]

Quantitative Adjointness:

\[
f \star g \preceq j \quad \text{iff} \quad f \preceq g \rightarrow_{\star} j
\]
Adjointness of → ⋆ and ⊗

Classical adjointness:

\[(F \star G) \Rightarrow J \Leftrightarrow F \Rightarrow (G \rightarrow J)\]

Quantitative Adjunction:

\[f \star g \preceq j \iff f \preceq g \rightarrow j\]

My personal Aha Erlebnis:

\[a - b \preceq c \iff a \preceq b + c\]
\[a \cdot b \preceq c \iff a \preceq c / b\]
Quantitative Separation Logic

Overview of QSL:

Conservative extension of classical separation
Weakest–precondition–style reasoning about probabilistic programs with pointers
Not possible with the “Quantitative Separation Logic” of [Bozga, Iosif, Perarnau’08]

Supports inductive definitions of quantities

Future Work on QSL:
A syntax for quantitative separation logic
Automation: Quantitative entailment? Quantitative symbolic heaps?
Cyclic proofs for inductive quantities?
Concurrency

---

Overview of QSL:

- Conservative extension of classical separation

---

2 Bozga, Iosif, Perarnau. Quantitative Separation Logic and Programs with Lists. IJCAR'08.
Overview of QSL:

- **Conservative extension** of classical separation
- **Weakest-precondition-style** reasoning about probabilistic programs with pointers

---

²Bozga, Iosif, Perarnau. Quantitative Separation Logic and Programs with Lists. IJCAR'08.
Overview of QSL:

- **Conservative extension** of classical separation
- **Weakest–precondition–style reasoning** about probabilistic programs with pointers
  - Not possible with the “Quantitative Separation Logic” of [Bozga, Iosif, Perarnau'08]²

²Bozga, Iosif, Perarnau. Quantitative Separation Logic and Programs with Lists. IJCAR'08.
Overview of QSL:

- **Conservative extension** of classical separation
- Weakest–precondition–style reasoning about probabilistic programs with pointers
  - Not possible with the “Quantitative Separation Logic” of [Bozga, Iosif, Perarnau’08]²
- Supports inductive definitions of quantities

---

²Bozga, Iosif, Perarnau. Quantitative Separation Logic and Programs with Lists. IJCAR'08.
Overview of QSL:

- **Conservative extension** of classical separation
- **Weakest-precondition-style reasoning** about probabilistic programs with pointers
  - Not possible with the “Quantitative Separation Logic” of [Bozga, Iosif, Perarnau'08]²
- **Supports inductive definitions of quantities**

Future Work on QSL:

---

²Bozga, Iosif, Perarnau. Quantitative Separation Logic and Programs with Lists. IJCAR'08.
Overview of QSL:

- **Conservative extension** of classical separation
- **Weakest–precondition–style reasoning about probabilistic programs with pointers**
  - Not possible with the “Quantitative Separation Logic” of [Bozga, Iosif, Perarnau'08]²
- **Supports inductive definitions of quantities**

Future Work on QSL:

- A **syntax** for quantitative separation logic

---

²Bozga, Iosif, Perarnau. Quantitative Separation Logic and Programs with Lists. IJCAR'08.
Overview of QSL:

- **Conservative extension** of classical separation
- **Weakest–precondition–style reasoning** about probabilistic programs with pointers
  - Not possible with the “Quantitative Separation Logic” of [Bozga, Iosif, Perarnau'08]²
- **Supports inductive definitions** of quantities

Future Work on QSL:

- A **syntax** for quantitative separation logic
- **Automation**: Quantitative entailment? Quantitative symbolic heaps?

---

²Bozga, Iosif, Perarnau. Quantitative Separation Logic and Programs with Lists. IJCAR'08.
Overview of QSL:

- **Conservative extension** of classical separation
- Weakest–precondition–style reasoning about probabilistic programs with pointers
  - Not possible with the “Quantitative Separation Logic” of [Bozga,Iosif,Perarnau’08]²
- **Supports inductive definitions of quantities**

Future Work on QSL:

- A **syntax** for quantitative separation logic
- **Automation:** Quantitative entailment? Quantitative symbolic heaps?
- **Cyclic proofs** for inductive quantities?
- **Concurrency**

² Bozga, Iosif, Perarnau. Quantitative Separation Logic and Programs with Lists. IJCAR'08.
Overview of QSL:

- **Conservative extension** of classical separation
- Weakest–precondition–style reasoning about probabilistic programs with pointers
  - Not possible with the “Quantitative Separation Logic” of [Bozga, Iosif, Perarnau’08]²
- **Supports** inductive definitions of quantities

Future Work on QSL:

- A **syntax** for quantitative separation logic
- Automation: Quantitative entailment? Quantitative symbolic heaps?
- Cyclic proofs for inductive quantities?
- Concurrency

---

²Bozga, Iosif, Perarnau. Quantitative Separation Logic and Programs with Lists. IJCAR’08.

Thank you for your kind attention!