Inferring Expected Runtimes for Probabilistic Integer Programs Using Expected Sizes

27th Conference on Tools and Algorithms for the Construction and Analysis of Systems

Fabian Meyer  Marcel Hark  Jürgen Giesl
Randomization in Programming

• Recently: growing interest in randomization in programming.
• Extension of classical programs by probability distributions.
  → efficiency of algorithms, cryptography, ...

```latex
while (x > 0) {
  x ← x − 1
  while (x) {
    x ← x − 1
  }
  \[ \frac{1}{2} \]
}
```

• Termination behavior diversifies.
  → Measure of efficiency: expected runtime.
  → How to infer upper bounds on expected runtime fully automatically?

Introduction

Randomization in Programming

• Recently: growing interest in randomization in programming.

• Extension of classical programs by probability distributions.

→ Efficiency of algorithms, cryptography, ...

• Termination behavior diversifies.

→ Measure of efficiency: expected runtime.

→ How to infer upper bounds on expected runtime fully automatically?
Introduction

Randomization in Programming

- Recently: growing interest in randomization in programming.
- Extension of classical programs by probability distributions.

→ Efficiency of algorithms, cryptography, ...

while \( x > 0 \)
\[
\begin{align*}
x & \leftarrow x - 1 \\
x & \leftarrow x - 1
\end{align*}
\]

- Termination behavior diversifies.

→ Measure of efficiency: expected runtime.

→ How to infer upper bounds on expected runtime fully automatically?
Randomization in Programming

- Recently: growing interest in randomization in programming.
- Extension of classical programs by probability distributions.
  \[ \text{→ efficiency of algorithms, cryptography, ...} \]
Randomization in Programming

- Recently: growing interest in randomization in programming.
- Extension of classical programs by probability distributions.
  → efficiency of algorithms, cryptography, …
Introduction

Randomization in Programming

- Recently: growing interest in randomization in programming.
- Extension of classical programs by probability distributions.
  → efficiency of algorithms, cryptography, ...
Introduction

Randomization in Programming

• Recently: growing interest in randomization in programming.
• Extension of classical programs by probability distributions.
  → efficiency of algorithms, cryptography, ...
• Termination behavior diversifies.

while\(x > 0\)\
{ \\
  x \leftarrow x \\ \\
  \begin{cases} \\
    \frac{1}{2} & \text{if } x > 0 \\
    x \leftarrow x - 1 & \text{else}
  \end{cases}
}
Introduction

Randomization in Programming

- Recently: growing interest in randomization in programming.
- Extension of classical programs by probability distributions.
  → efficiency of algorithms, cryptography, ... 
- Termination behavior diversifies.
  → Measure of efficiency: expected runtime.

```plaintext
while \( x > 0 \) {
  \{ x \leftarrow x \} \left[ \frac{1}{2} \right] \{ x \leftarrow x - 1 \}
}
```
Randomization in Programming

- Recently: growing interest in randomization in programming.
- Extension of classical programs by probability distributions.
  → efficiency of algorithms, cryptography, ...
- Termination behavior diversifies.
  → Measure of efficiency: expected runtime.
  → How to infer upper bounds on expected runtime fully automatically?

```plaintext
while(x > 0)
{
x ← x \left\lfloor \frac{\alpha}{2} \right\rfloor \{ x ← x - 1 \}
}
```
Introduction

Linear Probabilistic Ranking Functions (LPRF)

\[
\begin{aligned}
\text{while } (x > 0) \{ \quad \{ x \leftarrow x \} \quad \{ x \leftarrow x - 1 \} \\
\end{aligned}
\]

Consider \( r = 2 \cdot x \).
Whenever loop can be entered: \( r > 0 \).
In one loop iteration \( r \) is expected to decrease by 1 in each iteration.

\[
\frac{1}{2} \cdot r \left\lfloor \frac{x}{x} \right\rfloor + \frac{1}{2} \cdot r \left\lfloor \frac{x}{x-1} \right\rfloor = 2 \cdot x - 1 = r - 1
\]

Expected runtime of loop: at most \( r = 2 \cdot x \). (e.g., [Bournez & Garnier '05]).
Introduction

Linear Probabilistic Ranking Functions (LPRF)

- Automatic complexity analysis of classical programs:

```plaintext
while (x > 0)
    { x ← x }

1/2 \cdot r[x/x] + 1/2 \cdot r[x/x-1] = 2 \cdot x - 1 = r - 1

Expected runtime of loop: at most r = 2 \cdot x. (e.g., [Bournez & Garnier '05].)
```
Introduction

Linear Probabilistic Ranking Functions (LPRF)

• Automatic complexity analysis of classical programs:
  → Linear ranking functions.
Linear Probabilistic Ranking Functions (LPRF)

- Automatic complexity analysis of classical programs:
  → Linear ranking functions.
- In case of randomization: linear probabilistic ranking functions.
Introduction

Linear Probabilistic Ranking Functions (LPRF)

- Automatic complexity analysis of classical programs:
  → Linear ranking functions.
- In case of randomization: linear probabilistic ranking functions.

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

Consider \( r = 2 \cdot x \).
Whenever loop can be entered: \( r > 0 \).
In one loop iteration \( r \) is expected to decrease by 1 in each iteration.

\[
\frac{1}{2} \cdot r \left[ \frac{x}{x} \right] + \frac{1}{2} \cdot r \left[ \frac{x}{x} - 1 \right] = 2 \cdot x - 1 = r - 1
\]

Expected runtime of loop: at most \( r = 2 \cdot x \). (e.g., [Bournez & Garnier '05]).
Introduction

Linear Probabilistic Ranking Functions (LPRF)

- Automatic complexity analysis of classical programs:
  → Linear ranking functions.
- In case of randomization: linear probabilistic ranking functions.

Consider \( r = 2 \cdot x \).

```plaintext
while (x > 0) {
    \{ x \leftarrow x \} \left[ \frac{1}{2} \right] \{ x \leftarrow x - 1 \}
}
```
Linear Probabilistic Ranking Functions (LPRF)

- Automatic complexity analysis of classical programs:
  - Linear ranking functions.
- In case of randomization: linear probabilistic ranking functions.

Consider $\tau = 2 \cdot x$.
 Whenever loop can be entered: $\tau > 0$.

```latex
while(x > 0)\
{ x \leftarrow x \times \frac{1}{2} } \{ x \leftarrow x - 1 \}
```
Introduction

Linear Probabilistic Ranking Functions (LPRF)

- Automatic complexity analysis of classical programs:
  → Linear ranking functions.
- In case of randomization: linear probabilistic ranking functions.

Consider \( r = 2 \cdot x \).

Whenever loop can be entered: \( r > 0 \).

In one loop iteration \( r \) is expected to decrease by 1 in each iteration.

\[
\text{while}(x > 0) \{
\{ x \leftarrow x \} \left[ \frac{1}{2} \right] \{ x \leftarrow x - 1 \}
\}
\]
Introduction

Linear Probabilistic Ranking Functions (LPRF)

- Automatic complexity analysis of classical programs:
  \[ \rightarrow \text{Linear ranking functions.} \]
- In case of randomization: linear probabilistic ranking functions.

Consider \( r = 2 \cdot x \).

Whenever loop can be entered: \( r > 0 \).

In one loop iteration \( r \) is expected to decrease by 1 in each iteration.

\[
\frac{1}{2} \cdot r \left[ \frac{x}{x} \right] + \frac{1}{2} \cdot r \left[ \frac{x}{x - 1} \right] = 2 \cdot x - 1 = r - 1
\]

while \( (x > 0) \{
\{ x \leftarrow x \} \frac{1}{2} \{ x \leftarrow x - 1 \}
\} \)
Linear Probabilistic Ranking Functions (LPRF)

- Automatic complexity analysis of classical programs: → Linear ranking functions.
- In case of randomization: linear probabilistic ranking functions.

Consider $r = 2 \cdot x$.
Whenever loop can be entered: $r > 0$.
In one loop iteration $r$ is expected to decrease by 1 in each iteration.

\[
\frac{1}{2} \cdot r \frac{x}{x} + \frac{1}{2} \cdot r \left( \frac{x}{x} - 1 \right) = 2 \cdot x - 1 = r - 1
\]

Expected runtime of loop: at most $r = 2 \cdot x$. (e.g., [Bournez & Garnier '05]).
Introduction

Linear Probabilistic Ranking Functions (LPRF)

while \(x > 0\) {
\{ \begin{align*}
x & \leftarrow x \\
{\left[ \frac{1}{2} \right]} & 
\{ x \leftarrow x - 1 \}
\end{align*} \}
}

• What about larger programs?

while \(y > 0\) {
\begin{align*}
y & \leftarrow y - 1
\end{align*}
}

• Value of \(y\) grows in first loop.

→ Cannot bound expected runtime with single LPRF.

• Still, \(y\) is LPRF for the (standalone) second loop.

• Expected runtime of full program (intuitively):

→ \(2 \cdot x + "\text{expected size}"(y)\).

• Computation of runtimes via sizes:

→ very successful for classical programs [Giesl et al. ’16].

• Contribution: Novel modular approach by combining expected runtimes and expected sizes.
Linear Probabilistic Ranking Functions (LPRF)

• What about larger programs?

while (x > 0) {
    x ← x
    [1/2]
    x ← x − 1
}
Linear Probabilistic Ranking Functions (LPRF)

- What about larger programs?

```plaintext
while (x > 0) {
  \{ y ← y \} \quad \{ x ← x \} \quad \{ y ← y + x \} \quad \{ x ← x - 1 \}
}
while (y > 0) {
  y ← y - 1
}
```

- Value of \( y \) grows in first loop.

→ Cannot bound expected runtime with single LPRF.

- Still, \( y \) is LPRF for the (standalone) second loop.

- Expected runtime of full program (intuitively):

→ \( 2 \cdot x + \text{"expected size"}(y) \).

- Computation of runtimes via sizes:

→ very successful for classical programs [Giesl et al. ‘16].

- Contribution: Novel modular approach by combining expected runtimes and expected sizes.
Introduction

Linear Probabilistic Ranking Functions (LPRF)

- What about larger programs?
- Value of $y$ grows in first loop.

```plaintext
while (x > 0) {
    \{ y ← y \\
    x ← x \} \frac{1}{2} \{ y ← y + x \\
    x ← x - 1 \}
}

while (y > 0) {
    y ← y - 1
}
```

- Still, $y$ is LPRF for the (standalone) second loop.
- Expected runtime of full program (intuitively):
  $\rightarrow 2 \cdot x + \text{“expected size” (y)}$.

- Computation of runtimes via sizes:
  $\rightarrow$ very successful for classical programs [Giesl et al. ’16].

- Contribution: Novel modular approach by combining expected runtimes and expected sizes.
Introduction

Linear Probabilistic Ranking Functions (LPRF)

- What about larger programs?
- Value of $y$ grows in first loop.
  → Cannot bound expected runtime with single LPRF.

```plaintext
while (x > 0) {
  { y ← y
    x ← x
  } [\frac{1}{2}]
  { y ← y + x
    x ← x - 1
  }
}
while (y > 0) {
  y ← y - 1
}
```

• Value of $y$ grows in first loop.
  → Cannot bound expected runtime with single LPRF.

→ Cannot bound expected runtime with single LPRF.

• Still, $y$ is LPRF for the (standalone) second loop.

• Expected runtime of full program (intuitively):
  → $2 \cdot x + \text{"expected size" (y)}$.

• Computation of runtimes via sizes:
  → very successful for classical programs [Giesl et al. '16].

• Contribution: Novel modular approach by combining expected runtimes and expected sizes.
Introduction

Linear Probabilistic Ranking Functions (LPRF)

- What about larger programs?
- Value of $y$ grows in first loop.
  → Cannot bound expected runtime with single LPRF.
- Still, $y$ is LPRF for the (standalone) second loop.
Introduction

Linear Probabilistic Ranking Functions (LPRF)

- What about larger programs?
- Value of $y$ grows in first loop.
  → Cannot bound expected runtime with single LPRF.
- Still, $y$ is LPRF for the (standalone) second loop.
- Expected runtime of full program (intuitively):

\[
\begin{align*}
\text{while}(x > 0) & \{ \\
& \{ y \leftarrow y \} \ \{ x \leftarrow x \} \ \{ y \leftarrow y + x \} \ \{ x \leftarrow x - 1 \} \\
\text{while}(y > 0) & \{ \\
& y \leftarrow y - 1
\end{align*}
\]
Introduction

Linear Probabilistic Ranking Functions (LPRF)

- What about larger programs?
- Value of $y$ grows in first loop. 
  $\rightarrow$ Cannot bound expected runtime with single LPRF.
- Still, $y$ is LPRF for the (standalone) second loop.
- Expected runtime of full program (intuitively):
  $\rightarrow 2 \cdot x + \text{“expected size” } (y).$

\[
\begin{align*}
\text{while } (x > 0) \{ \\
\quad \begin{cases}
\quad y \leftarrow y \\
\quad x \leftarrow x
\end{cases} & \quad \frac{1}{2} \\
\end{align*}
\]

\[
\begin{align*}
\text{while } (y > 0) \{ \\
\quad y \leftarrow y - 1
\end{align*}
\]
Introduction

Linear Probabilistic Ranking Functions (LPRF)

- What about larger programs?
- Value of \( y \) grows in first loop.
  \( \rightarrow \) Cannot bound expected runtime with single LPRF.
- Still, \( y \) is LPRF for the (standalone) second loop.
- Expected runtime of full program (intuitively):
  \( \rightarrow 2 \cdot x + \text{“expected size”} \ (y) \).
- Computation of runtimes via sizes:

\[
\text{while}(x > 0)\
\begin{cases}
    y & \leftarrow y \\n    x & \leftarrow x \\
\end{cases}
\text{[1/2]}
\begin{cases}
    y & \leftarrow y + x \\n    x & \leftarrow x - 1 \\
\end{cases}
\]

\text{while}(y > 0)\
\begin{cases}
    y & \leftarrow y - 1 \\
\end{cases}
Introduction

Linear Probabilistic Ranking Functions (LPRF)

- What about larger programs?
- Value of \( y \) grows in first loop.
  \[ \text{→ Cannot bound expected runtime with single LPRF.} \]
- Still, \( y \) is LPRF for the (standalone) second loop.
- Expected runtime of full program (intuitively):
  \[ \text{→ } 2 \cdot x + \text{ “expected size” (} y \text{)} . \]
- Computation of runtimes via sizes:
  \[ \text{→ very successful for classical programs [Giesl et al. ’16].} \]

```
while ( x > 0 ) { 
{ \{ x ← x \} [1/2] \{ y ← y + x \} \{ x ← x − 1 \} 

while ( y > 0 ) { 
    y ← y − 1 
}
```
Introduction

Linear Probabilistic Ranking Functions (LPRF)

- What about larger programs?
- Value of \( y \) grows in first loop.
  \( \rightarrow \) Cannot bound expected runtime with single LPRF.
- Still, \( y \) is LPRF for the (standalone) second loop.
- Expected runtime of full program (intuitively):
  \( \rightarrow 2 \cdot x + \text{“expected size”} \ (y) \).
- Computation of runtimes via sizes:
  \( \rightarrow \) very successful for classical programs [Giesl et al. ’16].
- **Contribution**: Novel modular approach by combining expected runtimes and expected sizes.

```plaintext
while (x > 0) {
  \{ y ← y
  x ← x \} \[1/2\] \{ y ← y + x
  x ← x − 1 \}
}
while (y > 0) {
  y ← y − 1
}
```
Introduction

Probabilistic Integer Transition Systems
Introduction

Probabilistic Integer Transition Systems

- We denote programs by graphs to capture control flow.
Probabilistic Integer Transition Systems

- We denote programs by graphs to capture control flow.

```plaintext
while (x > 0) {
  \{ y ← y \\
  x ← x \} \begin{array}{l}
\end{array}
\begin{array}{l}
\begin{array}{l}
\end{array}
\end{array}
\begin{array}{l}
\end{array}
\begin{array}{l}
\begin{array}{l}
\end{array}
\end{array} \{ y ← y + x \\
  x ← x - 1 \}
}
while (y > 0) {
  y ← y - 1
}
```
Introduction

Probabilistic Integer Transition Systems

- We denote programs by graphs to capture control flow.

\[
\begin{align*}
\ell_0 & : t_0 \in g_0 \\
\frac{1}{2} : t_1 \in g_1 & \quad \text{if} (x > 0) \\
y & \leftarrow y \\
x & \leftarrow x \\
\ell_1 & \\
\frac{1}{2} : t_2 \in g_1 & \quad \text{if} (x > 0) \\
y & \leftarrow y + x \\
x & \leftarrow x - 1 \\
\ell_2 & : t_3 \in g_2 \quad \text{if} (x \leq 0) \\
\ell_3 & : t_4 \in g_3 \quad \text{if} (y > 0) \\
y & \leftarrow y - 1
\end{align*}
\]

\[
\ell_1: \text{while}(x > 0) \{ \\
\{ y \leftarrow y \} \quad \left[ \frac{1}{2} \right] \{ y \leftarrow y + x \} \\
\{ x \leftarrow x \} \\
\}
\]

\[
\ell_2: \text{while}(y > 0) \{ \\
\{ y \leftarrow y - 1 \} \\
\}
\]
Introduction

Outline

Introduction

Computing Expected Runtime Bounds

Computing Expected Size Bounds

Experiments

Conclusion
Computing Expected Runtime Bounds

Overall Idea

\begin{itemize}
  \item For each general transition \( g \in \text{GT} \):
    \[ R(g) = \text{number of executions of } g \text{ in run of full program}. \]
  \item \[ \sum_{g \in \text{GT}} E\left( R(g) \right) \] is expected runtime of full program.
  \item Over-approximate \( E\left( R(g) \right) \) for each \( g \in \text{GT} \).
  \item Contribution: Modular inference of bound on \( E\left( R(g) \right) \).
\end{itemize}
Computing Expected Runtime Bounds

Overall Idea

- For each general transition \( g \in \mathcal{GT} \):

\[
\begin{align*}
\frac{1}{2} : t_0 &\in g_0 \\
&\text{if } (x \leq 0) \\
t_0 &\in g_0
\end{align*}
\]

\[
\begin{align*}
\frac{1}{2} : t_1 &\in g_1 \\
&\text{if } (x > 0) \\
y &\leftarrow y \\
x &\leftarrow x - 1 \\
t_1 &\in g_1
\end{align*}
\]

\[
\begin{align*}
\frac{1}{2} : t_2 &\in g_1 \\
&\text{if } (x > 0) \\
y &\leftarrow y + x \\
x &\leftarrow x - 1 \\
t_2 &\in g_1
\end{align*}
\]

\[
\begin{align*}
t_3 &\in g_2 \\
&\text{if } (x \leq 0) \\
t_4 &\in g_3 \\
&\text{if } (y > 0) \\
y &\leftarrow y - 1
\end{align*}
\]
Computing Expected Runtime Bounds

Overall Idea

- For each general transition $g \in \mathcal{G}$:
  \[ R(g) = \text{number of executions of } g \text{ in run of full program.} \]

- Over-approximate $E(R(g))$ for each $g \in \mathcal{G}$.
- Contribution: Modular inference of bound on $E(R(g))$.

\[\begin{align*}
  \ell_0 & \quad t_0 \in g_0 \\
  \frac{1}{2} : t_1 \in g_1 & \quad \text{if } (x > 0) \\
  & \quad y \leftarrow y \\
  & \quad x \leftarrow x \\
  \frac{1}{2} : t_2 \in g_1 & \quad \text{if } (x > 0) \\
  & \quad y \leftarrow y + x \\
  & \quad x \leftarrow x - 1 \\
  \ell_1 & \\
  \ell_2 & \quad t_3 \in g_2 \quad \text{if } (x \leq 0) \\
  \frac{1}{2} : t_4 \in g_3 & \quad \text{if } (y > 0) \\
  & \quad y \leftarrow y - 1
\end{align*}\]
Computing Expected Runtime Bounds

Overall Idea

- For each general transition $g \in \mathcal{G}T$:
  
  $\mathcal{R}(g) = \text{number of executions of } g \text{ in run of full program.}$

- $\sum_{g \in \mathcal{G}T} \mathbb{E}(\mathcal{R}(g))$ is expected runtime of full program.

$\ell_0 \xrightarrow{t_0 \in \ell_0} \ell_1$

$\frac{1}{2} : t_1 \in g_1$
$\text{if } (x > 0)$
$y \leftarrow y$
$x \leftarrow x$

$\ell_1 \xrightarrow{t_1 \in \ell_1} \ell_2$

$\frac{1}{2} : t_2 \in g_1$
$\text{if } (x > 0)$
$y \leftarrow y + x$
$x \leftarrow x - 1$

$\ell_2 \xrightarrow{t_3 \in \ell_2} \ell_0$

$t_3 \in g_2$
$\text{if } (x \leq 0)$

$\ell_0 \xrightarrow{t_4 \in \ell_0} \ell_0$

$t_4 \in g_3$
$\text{if } (y > 0)$
$y \leftarrow y - 1$
Computing Expected Runtime Bounds

Overall Idea

- For each general transition $g \in \mathcal{G}$:
  $\mathcal{R}(g) =$ number of executions of $g$ in run of full program.
- $\sum_{g \in \mathcal{G}} \mathbb{E}(\mathcal{R}(g))$ is expected runtime of full program.
  $\rightarrow$ Over-approximate $\mathbb{E}(\mathcal{R}(g))$ for each $g \in \mathcal{G}$.

For each general transition $g \in \mathcal{G}$:

$R(g) =$ number of executions of $g$ in run of full program.

$\sum_{g \in \mathcal{G}} \mathbb{E}(\mathcal{R}(g))$ is expected runtime of full program.

$\rightarrow$ Over-approximate $\mathbb{E}(\mathcal{R}(g))$ for each $g \in \mathcal{G}$.

For each general transition $g \in \mathcal{G}$:

$R(g) =$ number of executions of $g$ in run of full program.

$\sum_{g \in \mathcal{G}} \mathbb{E}(\mathcal{R}(g))$ is expected runtime of full program.

$\rightarrow$ Over-approximate $\mathbb{E}(\mathcal{R}(g))$ for each $g \in \mathcal{G}$.
Computing Expected Runtime Bounds

Overall Idea

- For each general transition $g \in \mathcal{GT}$:
  \[ R(g) = \text{number of executions of } g \text{ in run of full program}. \]
- \[ \sum_{g \in \mathcal{GT}} \mathbb{E}(R(g)) \] is expected runtime of full program.
  → Over-approximate $\mathbb{E}(R(g))$ for each $g \in \mathcal{GT}$.
  → Contribution: Modular inference of bound on $\mathbb{E}(R(g))$.
Computing Expected Runtime Bounds

Overall Idea

\[ \ell_0 \]

\[ t_0 \in g_0 \]

\[ \ell_1 \]

\[ \frac{1}{2} : t_1 \in g_1 \]

\[ \text{if } (x > 0) \]

\[ y \leftarrow y \]

\[ x \leftarrow x - 1 \]

\[ \ell_2 \]

\[ t_3 \in g_2 \]

\[ \text{if } (x \leq 0) \]

\[ \ell_3 \]

\[ t_4 \in g_3 \]

\[ \text{if } (y > 0) \]

\[ y \leftarrow y - 1 \]

- How to over-approximate this expected value?

→ Expected value is not multiplicative.

\[ \frac{1}{2} : t_2 \in g_1 \]

\[ \text{if } (x > 0) \]

\[ y \leftarrow y + x \]

\[ x \leftarrow x \]

\[ t_2 \in g_1 \]
Computing Expected Runtime Bounds

Overall Idea

- Given sub-program $GT'$ of full program.
Computing Expected Runtime Bounds

**Overall Idea**

- Given sub-program $GT'$ of full program.

  Number of executions of $GT'$ in run of full program?

  
  $\frac{1}{2} : t_1 \in g_1$

  $\frac{1}{2} : t_2 \in g_1$

  $\text{if (} x > 0 \text{)}$

  $\text{if (} x > 0 \text{)}$

  $y \leftarrow y$

  $y \leftarrow y + x$

  $x \leftarrow x$

  $x \leftarrow x - 1$

  $t_3 \in g_2$

  $t_4 \in g_3$

  $\text{if (} x \leq 0 \text{)}$

  $\text{if (} y > 0 \text{)}$

  $y \leftarrow y - 1$

  $y \leftarrow y$
Computing Expected Runtime Bounds

Overall Idea

- Given sub-program $GT'$ of full program.
  Number of executions of $GT'$ in run of full program?
  $\leq \left( \# \text{ enter} (GT') \right) \cdot \left( \text{local-runtime}(GT')[v/"size"\text{enter}_{GT'}(v)] \right)$

\begin{align*}
\frac{1}{2} : t_1 & \in g_1 \\
\text{if } (x > 0) & \quad y \leftarrow y \\
& \quad x \leftarrow x
\end{align*}

\begin{align*}
\frac{1}{2} : t_2 & \in g_1 \\
\text{if } (x > 0) & \quad y \leftarrow y + x \\
& \quad x \leftarrow x - 1
\end{align*}

\begin{align*}
t_3 & \in g_2 \\
\text{if } (x \leq 0) & \\

t_4 & \in g_3 \\
\text{if } (y > 0) & \quad y \leftarrow y - 1
\end{align*}
Computing Expected Runtime Bounds

Overall Idea

- Given sub-program $GT'$ of full program.
  Number of executions of $GT'$ in run of full program?
  \[ \leq (\# \text{ enter } (GT')) \cdot (\text{local-runtime}(GT') [v/'size'_{\text{enter}} GT'(v)]) \]
  Expected number of executions of $GT'$?

\[
\begin{align*}
l_0 & \quad t_0 \in g_0 \\
\frac{1}{2} : l_1 & \quad t_1 \in g_1 \\
& \quad \text{if } (x > 0) \\
& \quad y \leftarrow y \\
& \quad x \leftarrow x \\
\frac{1}{2} : l_2 & \quad t_2 \in g_1 \\
& \quad \text{if } (x > 0) \\
& \quad y \leftarrow y + x \\
& \quad x \leftarrow x - 1 \\
\frac{1}{2} : l_0 & \\
& \quad t_3 \in g_2 \\
& \quad \text{if } (x \leq 0) \\
\frac{1}{2} : l_1 & \quad t_4 \in g_3 \\
& \quad \text{if } (y > 0) \\
& \quad y \leftarrow y - 1
\end{align*}
\]
Computing Expected Runtime Bounds

Overall Idea

- Given sub-program $GT'$ of full program.
  Number of executions of $GT'$ in run of full program?
  $$\leq (\# \text{enter}(GT')) \cdot (\text{local-runtime}(GT')[v/"size"_{enterGT'}(v)])$$
  Expected number of executions of $GT'$?
  $$\leq E\left(\left(\# \text{enter}(GT')\right) \cdot (\text{local-runtime}(GT')[v/"size"_{enterGT'}(v)])\right)$$
- How to over-approximate this expected value?
  → Expected value is not multiplicative.
Computing Expected Runtime Bounds

Overall Idea

- Given sub-program $GT'$ of full program.
  Number of executions of $GT'$ in run of full program?
  $\leq (\# \text{ enter } (GT')) \cdot (\text{local-runtime}(GT') [v/"size" \text{ enter}_{GT'}(v)])$
  Expected number of executions of $GT'$?
  $\leq E\left((\# \text{ enter } (GT')) \cdot (\text{local-runtime}(GT') [v/"size" \text{ enter}_{GT'}(v)])\right)$
- How to over-approximate this expected value?

→ Expected value is not multiplicative.
Computing Expected Runtime Bounds

Overall Idea

- Given sub-program $GT'$ of full program.
- Number of executions of $GT'$ in run of full program?
  \[ \leq \left( \# \text{ enter } (GT') \right) \cdot \left( \text{local-runtime}(GT') \left[ v/ \text{"size" enterGT'}(v) \right] \right) \]
- Expected number of executions of $GT'$?
  \[ \leq \mathbb{E}\left( \left( \# \text{ enter } (GT') \right) \cdot \left( \text{local-runtime}(GT') \left[ v/ \text{"size" enterGT'}(v) \right] \right) \right) \]

- How to over-approximate this expected value?
  → Expected value is not multiplicative.
Computing Expected Runtime Bounds

**Computation**

\[ E \left( \# \text{ enter } (GT') \cdot \text{local-runtime}(GT') [v/"size" \_\text{enterGT'}(v)] \right) \]

- If \( x \leq 0 \):
  - \( t_2 \in g_1 \)
  - \( y \leftarrow y + x \)
  - \( x \leftarrow x - 1 \)

- If \( x > 0 \):
  - \( y \leftarrow y \)
  - \( x \leftarrow x \)

- If \( y > 0 \):
  - \( t_4 \in g_3 \)
  - \( y \leftarrow y - 1 \)
Computing Expected Runtime Bounds

**Computation**

\[
\mathbb{E}\left(\# \text{ enter } (GT') \cdot (\text{local-runtime}(GT') [v/"size" \text{ enter}_{GT'}(v)])\right)
\]

- How to over-approximate this expected value?

1. \(t_0 \in g_0\)
   - \(\frac{1}{2}: t_1 \in g_1\)
   - if \((x > 0)\)
   - \(y \leftarrow y\)
   - \(x \leftarrow x\)

2. \(t_2 \in g_1\)
   - if \((x > 0)\)
   - \(y \leftarrow y + x\)
   - \(x \leftarrow x - 1\)

3. \(t_3 \in g_2\)
   - if \((x \leq 0)\)

4. \(t_4 \in g_3\)
   - if \((y > 0)\)
   - \(y \leftarrow y - 1\)
Computing Expected Runtime Bounds

Computation

\[ \mathbb{E} \left( \# \text{ enter (GT')} \right) \cdot \left( \text{local-runtime(GT')} \left[ v / \text{"size"}_{\text{enterGT'}} (v) \right] \right) \]

- How to over-approximate this expected value?

  \# enter (GT'):

  - Use (classical) worst-case bound from [Giesl et al. '16].
  - # enter (GT') = 1.
Computing Expected Runtime Bounds

Computation

\[
\mathbb{E}\left((\# \text{ enter } (GT')) \cdot (\text{local-runtime}(GT') [v/\text{"size"}_{\text{enter GT'}}(v)])\right)
\]

- How to over-approximate this expected value?

  \# enter (GT'):
  
  → Use (classical) worst-case bound from [Giesl et al. '16].

\[
\begin{align*}
\ell_0 & \rightarrow t_0 \in g_0 \\
\ell_1 & \rightarrow \frac{1}{2} : t_1 \in g_1 \\
& \quad \text{if } (x > 0) \\
& \quad y \leftarrow y \\
& \quad x \leftarrow x \\
\ell_2 & \rightarrow t_3 \in g_2 \\
& \quad \text{if } (x \leq 0) \\
\ell_3 & \rightarrow t_4 \in g_3 \\
& \quad \text{if } (y > 0) \\
& \quad y \leftarrow y - 1
\end{align*}
\]
Computing Expected Runtime Bounds

Computation

\[(\# \text{ enter } (\text{GT}') \land E \left( (\text{local-runtime}(\text{GT}') [v/"size"_{enter\text{GT}'}(v)]) \right) \]

- How to over-approximate this expected value?
  - \# enter (\text{GT}'):
    → Use (classical) worst-case bound from [Giesl et al. '16].

```plaintext
\begin{align*}
\ell_0 & \quad t_0 \in g_0 \\
\ell_1 & \quad \frac{1}{2} : t_1 \in g_1 \\
& \quad \text{if } (x > 0) \\
& \quad y \leftarrow y \\
& \quad x \leftarrow x \\
\ell_2 & \quad t_3 \in g_2 \\
& \quad \text{if } (x \leq 0) \\
\ell_3 & \quad t_4 \in g_3 \\
& \quad \text{if } (y > 0) \\
& \quad y \leftarrow y - 1
\end{align*}
```

Computing Expected Runtime Bounds

Computation

\[
(\# \text{ enter } (GT')) \cdot \mathbb{E}\left( (\text{local-runtime}(GT') [v/"size"_{\text{enterGT'}}(v)]) \right)
\]

- How to over-approximate this expected value?
  - \# enter (GT'):
    - Use (classical) worst-case bound from [Giesl et al. '16].

### Diagram

- \( t_0 \in g_0 \) if (\( x \leq 0 \))
  - \( \frac{1}{2} : t_1 \in g_1 \) if (\( x > 0 \))
    - \( y \leftarrow y \)
    - \( x \leftarrow x - 1 \)
- \( t_2 \in g_1 \) if (\( x > 0 \))
  - \( y \leftarrow y + x \)
  - \( x \leftarrow x - 1 \)
- \( t_3 \in g_2 \) if (\( x \leq 0 \))
- \( t_4 \in g_3 \) if (\( y > 0 \))
  - \( y \leftarrow y - 1 \)
Computing Expected Runtime Bounds

Computation

\[
\# \text{ enter } (GT') \cdot \mathbb{E} \left( \text{local-runtime}(GT') \left[ v/\text{"size"}_{\text{enter}GT'}(v) \right] \right)
\]

- How to over-approximate this expected value?
  - \# enter (GT'):
    - Use (classical) worst-case bound from [Giesl et al. '16].
    - \# enter (GT') = 1.
Computing Expected Runtime Bounds

Computation

$$\mathbb{E}\left(\text{local-runtime}(\text{GT'})[v/\text{"size"}_{\text{enter GT'}}(v)]\right)$$

- How to over-approximate this expected value?

Computing Expected Runtime Bounds

Computation

$$\mathbb{E}\left(\text{local-runtime}(GT') \left[v/\text{"size"}_{\text{enter}GT'}(v)]\right)\right)$$

- How to over-approximate this expected value?
  - Use linear (probabilistic) ranking function $r$ for $GT'$.

→ Already seen: $y$ is a ranking function for $GT'$.
→ Later: expected size of $y$ after $g_2$: $x^0 + y^0$. 

Computing Expected Runtime Bounds

Computation

\[ \tau \left[ v / \mathbb{E} \left( \text{"size"}_\text{enter}^{GT'}(v) \right) \right] \]

- How to over-approximate this expected value?
  - Use linear (probabilistic) ranking function \( \tau \) for \( GT' \).

\begin{align*}
& t_0 \in g_0 \\
& \frac{1}{2} : t_1 \in g_1 \\
& \text{if } (x > 0) \\
& y \leftarrow y \\
& x \leftarrow x \\
& \frac{1}{2} : t_2 \in g_1 \\
& \text{if } (x > 0) \\
& y \leftarrow y \leftarrow x + x \\
& x \leftarrow x - 1 \\
& t_3 \in g_2 \\
& \text{if } (x \leq 0) \\
& t_4 \in g_3 \\
& \text{if } (y > 0) \\
& y \leftarrow y - 1
\end{align*}
Computing Expected Runtime Bounds

Computation

\[ r \left[ v / \mathbb{E}(\text{"size"}_{\text{enterGT'}}(v)) \right] \]

- How to over-approximate this expected value?
  - Use linear (probabilistic) ranking function \( r \) for \( GT' \).
  - Already seen: \( y \) is a ranking function for \( GT' \).

Computing Expected Runtime Bounds

**Computation**

\[ y \left[ y / \mathbb{E}\left( \text{“size”}_{\text{enter}GT'}(y) \right) \right] \]

- How to over-approximate this expected value?
  - Use linear (probabilistic) ranking function \( r \) for \( GT' \).
  - Already seen: \( y \) is a ranking function for \( GT' \).

\[
\begin{align*}
&\ell_0 \\
&t_0 \in g_0 \\
&\frac{1}{2} : t_1 \in g_1 \\
&\quad \text{if } (x > 0) \\
&\quad y \leftarrow y \\
&\quad x \leftarrow x \\
&\frac{1}{2} : t_2 \in g_1 \\
&\quad \text{if } (x > 0) \\
&\quad y \leftarrow y + x \\
&\quad x \leftarrow x - 1 \\
&\ell_1 \\
&t_3 \in g_2 \\
&\quad \text{if } (x \leq 0) \\
&t_4 \in g_3 \\
&\quad \text{if } (y > 0) \\
&\quad y \leftarrow y - 1 \\
&\ell_2
\end{align*}
\]
Computing Expected Runtime Bounds

Computation

\[ y \left[ y / \mathbb{E}\left( \text{“size”}_{\text{enterGT'}}(y) \right) \right] \]

- How to over-approximate this expected value?
  - Use linear (probabilistic) ranking function \( r \) for \( GT' \).
  - Already seen: \( y \) is a ranking function for \( GT' \).
  - Later: expected size of \( y \) after \( g_2 \): \( x_0^2 + y_0 \).
Computing Expected Runtime Bounds

Computation

\[ x_0^2 + y_0 \]

- How to over-approximate this expected value?
  → Use linear (probabilistic) ranking function \( r \) for \( GT' \).
  → Already seen: \( y \) is a ranking function for \( GT' \).
  → Later: expected size of \( y \) after \( g_2 \): \( x_0^2 + y_0 \).
Computing Expected Runtime Bounds

Summary

\[ t_0 \in g_0 \]
\[ \frac{1}{2} : t_1 \in g_1 \]
\[ \text{if } (x > 0) \]
\[ y \leftarrow y \]
\[ x \leftarrow x \]
\[ \frac{1}{2} : t_2 \in g_1 \]
\[ \text{if } (x > 0) \]
\[ y \leftarrow y + x \]
\[ x \leftarrow x - 1 \]
\[ t_3 \in g_2 \]
\[ \text{if } (x \leq 0) \]
\[ t_4 \in g_3 \]
\[ \text{if } (y > 0) \]
\[ y \leftarrow y - 1 \]

Overall expected runtime of program:

\[ 1 + 2 \cdot x_0 + 1 + x_2 + y_0 \]

Implementation: heuristic chooses sub-programs.

• Modular approach for expected runtimes:
  → Use of (classical) worst-case runtime bounds.
  → Adaption of probabilistic ranking functions.
  → Use of expected sizes.
  → Alternating computation of runtime and size bounds.
Computing Expected Runtime Bounds

Summary

Overall expected runtime of program:

\[
\text{Overall expected runtime of program:} \\
\frac{1}{2} : t_1 \in g_1 \\
\text{if } (x > 0) \\
y \leftarrow y \\
x \leftarrow x \\
\frac{1}{2} : t_2 \in g_1 \\
\text{if } (x > 0) \\
y \leftarrow y + x \\
x \leftarrow x - 1 \\
t_3 \in g_2 \\
\text{if } (x \leq 0) \\
t_4 \in g_3 \\
\text{if } (y > 0) \\
y \leftarrow y - 1
\]
Computing Expected Runtime Bounds

Summary

Overall expected runtime of program:

\[ 1 + 2 \cdot x_0 + 1 + x_0^2 + y_0 \]
Computing Expected Runtime Bounds

Summary

Overall expected runtime of program:

\[ 1 + 2 \cdot x_0 + 1 + x_0^2 + y_0 \]

- Implementation: heuristic chooses sub-programs.
Computing Expected Runtime Bounds

Summary

Overall expected runtime of program:
\[ 1 + 2 \cdot x_0 + 1 + x_0^2 + y_0 \]

- Implementation: heuristic chooses sub-programs.
- **Modular** approach for expected runtimes:

\[ t_0 \in g_0 \]
\[ \frac{1}{2} : t_1 \in g_1 \quad \text{if} \ (x > 0) \]
\[ y \leftarrow y \]
\[ x \leftarrow x \]

\[ \frac{1}{2} : t_2 \in g_1 \quad \text{if} \ (x > 0) \]
\[ y \leftarrow y \]
\[ x \leftarrow x - 1 \]

\[ t_3 \in g_2 \quad \text{if} \ (x \leq 0) \]

\[ t_4 \in g_3 \quad \text{if} \ (y > 0) \]
\[ y \leftarrow y - 1 \]
Computing Expected Runtime Bounds

Summary

Overall expected runtime of program:

\[ 1 + 2 \cdot x_0 + 1 + x_0^2 + y_0 \]

- Implementation: heuristic chooses sub-programs.
- **Modular** approach for expected runtimes:
  - Use of (classical) worst-case runtime bounds.
  - Adaptation of probabilistic ranking functions.
  - Use of expected sizes.
  - Alternating computation of runtime and size bounds.
Computing Expected Runtime Bounds

Summary

Overall expected runtime of program:
\[1 + 2 \cdot x_0 + 1 + x_0^2 + y_0\]

- Implementation: heuristic chooses sub-programs.
- **Modular** approach for expected runtimes:
  - Use of (classical) worst-case runtime bounds.
  - Adaptation of probabilistic ranking functions.
  - Use of expected sizes.
  - Alternating computation of runtime and size bounds.

Computing Expected Runtime Bounds

Summary

Overall expected runtime of program:

\[ 1 + 2 \cdot x_0 + 1 + x_0^2 + y_0 \]

- Implementation: heuristic chooses sub-programs.
- **Modular** approach for expected runtimes:
  → Use of (classical) worst-case runtime bounds.
  → Adaptation of probabilistic ranking functions.
  → Use of expected sizes.
Computing Expected Runtime Bounds

Summary

- Overall expected runtime of program:
  \[ 1 + 2 \cdot x_0 + 1 + x_0^2 + y_0 \]
- Implementation: heuristic chooses sub-programs.
- **Modular** approach for expected runtimes:
  → Use of (classical) worst-case runtime bounds.
  → Adaption of probabilistic ranking functions.
  → Use of expected sizes.
  → Alternating computation of runtime and size bounds.
Computing Expected Size Bounds

Overall Idea

---

What means "size"?
→ \( S(g, v) \): largest value \( v \) takes after execution of \( g \)
\leq \text{incoming-size}(v) + R(g) \cdot \text{worst-case-change}(g, v) \)
Computing Expected Size Bounds

Overall Idea

\[ t_0 \in g_0 \]
\[ \frac{1}{2} : t_2 \in g_1 \quad \text{if } (x > 0) \]
\[ y \leftarrow y \]
\[ x \leftarrow x - 1 \]
\[ t_3 \in g_2 \quad \text{if } (x \leq 0) \]
\[ \frac{1}{2} : t_2 \in g_1 \quad \text{if } (x > 0) \]
\[ y \leftarrow y + x \]
\[ x \leftarrow x - 1 \]
\[ t_4 \in g_3 \quad \text{if } (y > 0) \]
\[ y \leftarrow y - 1 \]

What means "size"?
\[ \rightarrow S(g, v) : \text{largest value } v \text{ takes after execution of } g \]
\[ \leq \text{ incoming-size}(v) + R(g) \cdot \text{worst-case-change}(g, v) \]
Computing Expected Size Bounds

Overall Idea

• What means “size”?

\[
\frac{1}{2} : t_2 \in g_1 \\
\text{if } (x > 0) \\
y \leftarrow y \\
x \leftarrow x
\]

\[
\frac{1}{2} : t_2 \in g_1 \\
\text{if } (x > 0) \\
y \leftarrow y + x \\
x \leftarrow x - 1
\]

\[
t_3 \in g_2 \\
\text{if } (x \leq 0)
\]

\[
t_4 \in g_3 \\
\text{if } (y > 0) \\
y \leftarrow y - 1
\]
Computing Expected Size Bounds

Overall Idea

- What means “size”? → \( S(g, v) \): largest value \( v \) takes after execution of \( g \)

\[
\begin{align*}
\text{if } (x \leq 0) & \quad t_3 \in g_2 \\
\text{if } (x > 0) & \quad t_2 \in g_1
\end{align*}
\]

\[
\begin{align*}
y & \leftarrow y \\
x & \leftarrow x
\end{align*}
\]

\[
\begin{align*}
\frac{1}{2} & : t_2 \in g_1 \\
\text{if } (x > 0) & \\
y & \leftarrow y + x \\
x & \leftarrow x - 1
\end{align*}
\]

\[
\begin{align*}
t_4 \in g_3 \\
\text{if } (y > 0) & \\
y & \leftarrow y - 1
\end{align*}
\]
Computing Expected Size Bounds

Overall Idea

- What means “size”?
  - $S(g, v)$: largest value $v$ takes after execution of $g$
  - $\leq$ incoming-size($v$) + $R(g) \cdot$ worst-case-change($g, v$)

$$t_0 \in g_0$$

$$\frac{1}{2} : t_2 \in g_1$$
if ($x > 0$)

$$y \leftarrow y$$
$$x \leftarrow x$$

$$\frac{1}{2} : t_2 \in g_1$$
if ($x > 0$)

$$y \leftarrow y + x$$
$$x \leftarrow x - 1$$

$$t_3 \in g_2$$
if ($x \leq 0$)

$$t_4 \in g_3$$
if ($y > 0$)

$$y \leftarrow y - 1$$
Computing Expected Size Bounds

Overall Idea

- What means “size”?
  - \( S(g_1, y) \) largest value \( y \) takes after execution of \( g_1 \)
  \[ \leq \text{incoming-size}(y) + R(g_1) \cdot \text{worst-case-change}(g_1, y) \]

\[ \ell_0 \]
\[ t_0 \in g_0 \]

\[ \ell_1 \]
\[ \frac{1}{2} : t_2 \in g_1 \]
\[ \text{if } (x > 0) \]
\[ y \leftarrow y \]
\[ x \leftarrow x \]

\[ \ell_2 \]
\[ t_3 \in g_2 \]
\[ \text{if } (x \leq 0) \]

\[ \frac{1}{2} : t_2 \in g_1 \]
\[ \text{if } (x > 0) \]
\[ y \leftarrow y + x \]
\[ x \leftarrow x - 1 \]

\[ t_4 \in g_3 \]
\[ \text{if } (y > 0) \]
\[ y \leftarrow y - 1 \]

Computing Expected Size Bounds

Overall Idea

Expected size $\mathbb{E}(S(g_1, y))$:

$t_0 \in g_0$

$\frac{1}{2} : t_2 \in g_1$

if $(x > 0)$

$y \leftarrow y$

$x \leftarrow x$

$\frac{1}{2} : t_2 \in g_1$

if $(x > 0)$

$y \leftarrow y + x$

$x \leftarrow x - 1$

$t_3 \in g_2$

if $(x \leq 0)$

$t_4 \in g_3$

if $(y > 0)$

$y \leftarrow y - 1$
Computing Expected Size Bounds

Overall Idea

Expected size $\mathbb{E}(S(g_1, y))$:

$$\leq \mathbb{E}(\text{incoming-size}(y) + \mathcal{R}(g_1) \cdot \text{worst-case-change}(g_1, y))$$
Computing Expected Size Bounds

Overall Idea

Expected size $\mathbb{E}(S(g_1, y))$:

\[
\begin{align*}
\leq & \quad \mathbb{E}\left(\text{incoming-size}(y) + \mathcal{R}(g_1) \cdot \text{worst-case-change}(g_1, y)\right) \\
= & \quad \mathbb{E}\left(\text{incoming-size}(y)\right) + \mathbb{E}\left(\mathcal{R}(g_1) \cdot \text{worst-case-change}(g_1, y)\right)
\end{align*}
\]

$\ell_0$

$t_0 \in g_0$

$\frac{1}{2}: t_2 \in g_1$

if $(x > 0)$

$y \leftarrow y$

$x \leftarrow x - 1$

$\ell_1$

$t_3 \in g_2$

if $(x \leq 0)$

$t_4 \in g_3$

if $(y > 0)$

$y \leftarrow y - 1$

$\ell_2$

$t_1$

$\ell_3$

$t_2 \in g_1$

if $(x > 0)$

$y \leftarrow y + x$

$\frac{1}{2}$

What means “size”?

$\mathbb{E}(S(g_1, y))$: largest value $y$ takes after execution of $g_1$. $\mathbb{E}(\text{incoming-size}(y))$: expected incoming size $x$. $\mathbb{E}(\mathcal{R}(g_1) \cdot \text{worst-case-change}(g_1, y))$: expected worst-case change $y$. $\mathbb{E}(\text{incoming-size}(y)) + \mathbb{E}(\mathcal{R}(g_1) \cdot \text{worst-case-change}(g_1, y))$: expected size $y$. $\mathbb{E}(\text{incoming-size}(y)) + \mathbb{E}(\mathcal{R}(g_1) \cdot \text{worst-case-change}(g_1, y)) = y_0 + \mathbb{E}(\mathcal{R}(g_1) \cdot \text{worst-case-change}(g_1, y))$.
Computing Expected Size Bounds

Overall Idea

Expected size $\mathbb{E}(S(g_1, y))$:

$\leq \mathbb{E}(\text{incoming-size}(y) + \mathcal{R}(g_1) \cdot \text{worst-case-change}(g_1, y))$

$= \mathbb{E}(\text{incoming-size}(y)) + \mathbb{E}(\mathcal{R}(g_1) \cdot \text{worst-case-change}(g_1, y))$

What means "size"?

→ $S(g_1, y)$ largest value $y$ takes after execution of $g_1$

$\leq \text{incoming-size}(y) + \mathcal{R}(g_1) \cdot \text{worst-case-change}(g_1, y)$

$\mathbb{E}(\text{incoming-size}(y)) + \mathbb{E}(\mathcal{R}(g_1) \cdot \text{worst-case-change}(g_1, y))$

$= \mathbb{E}(\text{incoming-size}(y)) + \mathbb{E}(\mathcal{R}(g_1) \cdot \text{worst-case-change}(g_1, y))$

$= y_0 + \mathbb{E}(\mathcal{R}(g_1) \cdot \text{worst-case-change}(g_1, y))$
Computing Expected Size Bounds

Overall Idea

Expected size $\mathbb{E}(S(g_1, y))$:

\[
\leq \mathbb{E}(\text{incoming-size}(y) + R(g_1) \cdot \text{worst-case-change}(g_1, y))
\]

\[
= y_0 + \mathbb{E}(R(g_1) \cdot \text{worst-case-change}(g_1, y))
\]

\[
= \mathbb{E}(\text{incoming-size}(y)) + \mathbb{E}(R(g_1) \cdot \text{worst-case-change}(g_1, y))
\]

\[
= y_0 + \mathbb{E}(R(g_1) \cdot \text{worst-case-change}(g_1, y))
\]
Computing Expected Size Bounds

Computation

Expected size $\mathbb{E}(S(g_1, y))$:
$$\leq y_0 + \mathbb{E}(R(g_1) \cdot \text{worst-case-change}(g_1, y))$$
Computing Expected Size Bounds

Computation

Expected size $\mathbb{E}(S(g_1, y))$:

$\leq y_0 + \mathbb{E}(R(g_1) \cdot \text{worst-case-change}(g_1, y))$

$\rightarrow$ worst-case-change is independent of runtime.

$E(\text{worst-case-change}(g_1, y)) = y_0 + 2 \cdot x_0 \cdot x_0$

$= y_0 + x_0^2$
Computing Expected Size Bounds

Computation

Expected size $\mathbb{E}(S(g_1, y))$:
\[
\leq y_0 + \mathbb{E}(R(g_1) \cdot \text{worst-case-change}(g_1, y))
\]
Computing Expected Size Bounds

Computation

Expected size $\mathbb{E}(S(g_1,y))$:
\[
\leq y_0 + \mathbb{E}(\mathcal{R}(g_1) \cdot \text{worst-case-change}(g_1,y)) \\
= y_0 + \mathbb{E}(\mathcal{R}(g_1)) \cdot \mathbb{E}(\text{worst-case-change}(g_1,y))
\]
### Computation

Expected size $E(S(g_1, y))$:

- $\leq y_0 + E(R(g_1) \cdot \text{worst-case-change}(g_1, y))$
- $= y_0 + 2 \cdot x_0 \cdot E(\text{worst-case-change}(g_1, y))$

Diagram:

- $t_0 \in g_0$
  - $\frac{1}{2}: t_2 \in g_1 \quad \text{if } (x > 0)$
  - $y \leftarrow y$
  - $x \leftarrow x$
- $t_3 \in g_2 \quad \text{if } (x \leq 0)$
  - $t_4 \in g_3 \quad \text{if } (y > 0)$
  - $y \leftarrow y - 1$
- $t_1 \in g_1$
Computing Expected Size Bounds

Computation

Expected size $\mathbb{E}(S(g_1, y))$:

\[
\begin{align*}
\leq y_0 + \mathbb{E}(\mathcal{R}(g_1) \cdot \text{worst-case-change}(g_1, y)) \\
= y_0 + 2 \cdot x_0 \cdot \mathbb{E}(\text{worst-case-change}(g_1, y))
\end{align*}
\]

\[
\begin{align*}
t_0 & \in g_0 \\
\frac{1}{2} : t_2 & \in g_1 \quad \text{if } (x > 0) \\
& \quad y \leftarrow y + x \\
& \quad x \leftarrow x - 1 \\
\frac{1}{2} : t_2 & \in g_1 \quad \text{if } (x > 0) \\
& \quad y \leftarrow y + x \\
& \quad x \leftarrow x - 1 \\
t_3 & \in g_2 \quad \text{if } (x \leq 0) \\
t_4 & \in g_3 \quad \text{if } (y > 0) \\
& \quad y \leftarrow y - 1
\end{align*}
\]
Computing Expected Size Bounds

Computation

Expected size $\mathbb{E}(S(g_1, y))$:

\[
\begin{align*}
&\leq y_0 + \mathbb{E}(R(g_1) \cdot \text{worst-case-change}(g_1, y)) \\
&= y_0 + 2 \cdot x_0 \cdot \frac{x_0}{2}
\end{align*}
\]
Computing Expected Size Bounds

Computation

Expected size $\mathbb{E}(S(g_1, y))$:

<table>
<thead>
<tr>
<th>Condition</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x \leq 0$</td>
<td>$y \leftarrow y + x$</td>
</tr>
<tr>
<td>$x &gt; 0$</td>
<td>$y \leftarrow y + x$</td>
</tr>
<tr>
<td></td>
<td>$x \leftarrow x - 1$</td>
</tr>
</tbody>
</table>

$\mathbb{E}(S(g_1, y)) \leq y_0 + \mathbb{E}(R(g_1) \cdot \text{worst-case-change}(g_1, y))$

$= y_0 + x_0^2$

$\frac{1}{2} : t_2 \in g_1$

$\frac{1}{2} : t_2 \in g_1$

$\mathbb{E}(\text{worst-case-change}(g_1, y))$

$\mathbb{E}(R(g_1) \cdot \mathbb{E}(\text{worst-case-change}(g_1, y)))$

$= y_0 + 2 \cdot x_0 \cdot \mathbb{E}(\text{worst-case-change}(g_1, y))$

$= y_0 + 2 \cdot x_0 \cdot x_0^2$

$= y_0 + x_0^2$
Computing Expected Size Bounds

Computation

\[ t_0 \in g_0 \]
\[ \frac{1}{2} : t_2 \in g_1 \quad \text{if } (x > 0) \]
\[ y \leftarrow y \]
\[ x \leftarrow x \]
\[ \frac{1}{2} : t_2 \in g_1 \quad \text{if } (x > 0) \]
\[ y \leftarrow y + x \]
\[ x \leftarrow x - 1 \]
\[ t_3 \in g_2 \quad \text{if } (x \leq 0) \]
\[ t_4 \in g_3 \quad \text{if } (y > 0) \]
\[ y \leftarrow y - 1 \]
Computing Expected Size Bounds

Computation

\[
\begin{align*}
\ell_0 & \rightarrow t_0 \in g_0 \\
& \quad \frac{1}{2} : t_2 \in g_1 \quad \text{if} (x > 0) \\
& \quad y \leftarrow y \\
& \quad x \leftarrow x - 1 \\
\ell_1 & \rightarrow t_2 \in g_1 \quad \text{if} (x > 0) \\
& \quad y \leftarrow y + x \\
\ell_2 & \rightarrow t_3 \in g_2 \quad \text{if} (x \leq 0) \\
& \quad y \leftarrow y - 1 \\
\ell_3 & \rightarrow t_4 \in g_3 \quad \text{if} (y > 0) \\
& \quad y \leftarrow y - 1
\end{align*}
\]

Expected size of $v$ after $g_2$: $\leq$ Maximal expected size of $v$ after $g_0, g_1$.

→ Expected size of $y$ after $g_2$: $y_0 + x^2_0$.
Computing Expected Size Bounds

Computation

Expected size of $v$ after $g_2$:

- $t_0 \in g_0$
- $\frac{1}{2} : t_2 \in g_1$ if $(x > 0)$
- $y \leftarrow y$
- $x \leftarrow x$
- $\frac{1}{2} : t_2 \in g_1$ if $(x > 0)$
- $y \leftarrow y + x$
- $x \leftarrow x - 1$
- $t_3 \in g_2$ if $(x \leq 0)$
- $t_4 \in g_3$ if $(y > 0)$
- $y \leftarrow y - 1$
Computing Expected Size Bounds

Computation

Expected size of \( v \) after \( g_2 \):

\[ \leq \text{Maximal expected size of } v \text{ after } g_0, g_1. \]
Computing Expected Size Bounds

Computation

Expected size of $v$ after $g_2$:

$\leq$ Maximal expected size of $v$ after $g_0$, $g_1$.

$\rightarrow$ Expected size of $y$ after $g_2$: $y_0 + x_0^2$.
Computing Expected Size Bounds

Summary

\[ t_0 \in g_0 \]

\[ \frac{1}{2} : t_2 \in g_1 \]
if \( x > 0 \)
\[ y \leftarrow y \]
\[ x \leftarrow x - 1 \]

\[ t_3 \in g_2 \]
if \( x \leq 0 \)
\[ y \leftarrow y - 1 \]

\[ \frac{1}{2} : t_2 \in g_1 \]
if \( x > 0 \)
\[ y \leftarrow y + x \]
\[ x \leftarrow x - 1 \]

\[ t_4 \in g_3 \]
if \( y > 0 \)
\[ y \leftarrow y - 1 \]
Computing Expected Size Bounds

Summary

- Automatic inference of expected sizes:

\[ \begin{align*}
\ell_0 & \quad t_0 \in g_0 \\
\frac{1}{2} : t_2 \in g_1 & \quad \text{if } (x > 0) \\
y & \leftarrow y \\
x & \leftarrow x \\
\frac{1}{2} : t_2 \in g_1 & \quad \text{if } (x > 0) \\
y & \leftarrow y + x \\
x & \leftarrow x - 1 \\
t_3 \in g_2 & \quad \text{if } (x \leq 0) \\
t_4 \in g_3 & \quad \text{if } (y > 0) \\
y & \leftarrow y - 1
\end{align*} \]
Computing Expected Size Bounds

Summary

- Automatic inference of expected sizes:
  → Incoming expected sizes.

\[
\begin{align*}
\ell_0 & : t_0 \in g_0 \\
\frac{1}{2} & : t_2 \in g_1 \\
& \text{if } (x > 0) \\
y & \leftarrow y \\
x & \leftarrow x - 1 \\
\ell_1 & : t_2 \in g_1 \\
& \text{if } (x > 0) \\
y & \leftarrow y + x \\
x & \leftarrow x - 1 \\
\ell_2 & : t_3 \in g_2 \\
& \text{if } (x \leq 0) \\
y & \leftarrow y - 1 \\
\ell_3 & : t_4 \in g_3 \\
& \text{if } (y > 0) \\
y & \leftarrow y - 1
\end{align*}
\]
Computing Expected Size Bounds

Summary

- Automatic inference of expected sizes:
  - → Incoming expected sizes.
  - → Worst-case expected change.

```
\ell_0 \quad t_0 \in g_0
\frac{1}{2} : t_2 \in g_1 \quad \text{if } (x > 0)
\quad y \leftarrow y
\quad x \leftarrow x
\frac{1}{2} : t_2 \in g_1 \quad \text{if } (x > 0)
\quad y \leftarrow y + x
\quad x \leftarrow x - 1
\ell_1
\ell_2 \quad t_3 \in g_2 \quad \text{if } (x \leq 0)
\ell_3
\ell_4 \quad t_4 \in g_3 \quad \text{if } (y > 0)
\quad y \leftarrow y - 1
```

• Automatic inference of expected sizes:
  → Incoming expected sizes.
  → Worst-case expected change.
Computing Expected Size Bounds

Summary

- Automatic inference of expected sizes:
  - → Incoming expected sizes.
  - → Worst-case expected change.
  - → Expected runtime.

\[
\begin{align*}
\ell_0 &
\Rightarrow t_0 \in g_0 \\
\ell_1 &
\Rightarrow \begin{cases}
\frac{1}{2} : t_2 \in g_1 \\
\text{if } (x > 0) \\
y \leftarrow y + x \\
x \leftarrow x - 1
\end{cases} \\
\ell_2 &
\Rightarrow \begin{cases}
\frac{1}{2} : t_2 \in g_1 \\
\text{if } (x > 0) \\
y \leftarrow y + x \\
x \leftarrow x - 1
\end{cases} \\
\ell_3 &
\Rightarrow t_3 \in g_2 \\
\text{if } (x \leq 0) \\
y \leftarrow y - 1
\end{align*}
\]
Computing Expected Size Bounds

Summary

- Automatic inference of expected sizes:
  - → Incoming expected sizes.
  - → Worst-case expected change.
  - → Expected runtime.
- Implementation:
  - Handles transitions in topological order.
  - Graph to detect variables influencing expected change.
  - Uses classical worst-case sizes for these variables [Giesl et al. '16].
  - Combination: worst-case expected change.
Summary

- Automatic inference of expected sizes:
  - Incoming expected sizes.
  - Worst-case expected change.
  - Expected runtime.

- Implementation:
  - Handles transitions in topological order.

\[
\begin{align*}
\ell_0 & : t_0 \in g_0 \\
\ell_1 & : t_1 \in g_1 \\
\ell_2 & : t_2 \in g_2 \\
\ell_3 & : t_3 \in g_3 \\
\end{align*}
\]
Computing Expected Size Bounds

Summary

- Automatic inference of expected sizes:
  - Incoming expected sizes.
  - Worst-case expected change.
  - Expected runtime.
- Implementation:
  - Handles transitions in topological order.
  - Graph to detect variables influencing expected change.
Computing Expected Size Bounds

Summary

- Automatic inference of expected sizes:
  - → Incoming expected sizes.
  - → Worst-case expected change.
  - → Expected runtime.
- Implementation:
  - → Handles transitions in topological order.
  - → Graph to detect variables influencing expected change.
  - → Uses classical worst-case sizes for these variables [Giesl et al. '16].

$\ell_0 \ni t_0 \in g_0$

\[ \frac{1}{2} : t_2 \in g_1 \quad \text{if} \ (x > 0) \]
\[ y \leftarrow y \]
\[ x \leftarrow x \]

\[ \frac{1}{2} : t_2 \in g_1 \quad \text{if} \ (x > 0) \]
\[ y \leftarrow y + x \]
\[ x \leftarrow x - 1 \]

\[ t_3 \in g_2 \quad \text{if} \ (x \leq 0) \]

\[ t_4 \in g_3 \quad \text{if} \ (y > 0) \]
\[ y \leftarrow y - 1 \]
Experiments

Implementation
Experiments

Implementation

• Approach is implemented in KoAT [Giesl et al. ’16].
Experiments

Implementation

- Approach is implemented in KoAT [Giesl et al. '16].
- Uses SMT-solver Z3 [de Moura & Bjørner '08] and abstract domain library Apron [Jeannet & Mine '09].
Experiments

Implementation

- Approach is implemented in KoAT [Giesl et al. '16].
- Uses SMT-solver Z3 [de Moura & Bjørner '08] and abstract domain library Apron [Jeannet & Mine '09].
- Code is open-source, available via Github.
Experiments

Implementation

• Approach is implemented in KoAT [Giesl et al. ’16].
• Uses SMT-solver Z3 [de Moura & Bjørner ’08] and abstract domain library Apron [Jeannet & Mine ’09].
• Code is open-source, available via Github.
• Provide web interface, Docker image, static binary.
Experiments

Evaluation
Experiments

Evaluation

• Comparison with existing tools Absynth [Ngo et al. '18] and eco-imp [Avanzini et al. '20].
Experiments

Evaluation

- Comparison with existing tools Absynth [Ngo et al. '18] and eco-imp [Avanzini et al. '20].
- All 46 benchmarks from [Ngo et al. '18].
Experiments

Evaluation

- Comparison with existing tools Absynth [Ngo et al. '18] and eco-imp [Avanzini et al. '20].
- All 46 benchmarks from [Ngo et al. '18].
- 29 new examples containing 10 large examples from TPDB enriched with randomization.
Experiments

Evaluation

- Comparison with existing tools Absynth [Ngo et al. '18] and eco-imp [Avanzini et al. '20].
- All 46 benchmarks from [Ngo et al. '18].
- 29 new examples containing 10 large examples from TPDB enriched with randomization.
- Applied timeout of 5 minutes.
Experiments

Evaluation

## Experiments

### Evaluation

<table>
<thead>
<tr>
<th>Bound</th>
<th>KoAT</th>
<th>Absynth</th>
<th>eco-imp</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{O}(1)$</td>
<td>8</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>$\mathcal{O}(n)$</td>
<td>42</td>
<td>35</td>
<td>35</td>
</tr>
<tr>
<td>$\mathcal{O}(n^2)$</td>
<td>15</td>
<td>9</td>
<td>15</td>
</tr>
<tr>
<td>$\mathcal{O}(n^{2-2})$</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>EXP</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$\infty$</td>
<td>7</td>
<td>15</td>
<td>14</td>
</tr>
<tr>
<td>TO</td>
<td>0</td>
<td>9</td>
<td>3</td>
</tr>
</tbody>
</table>
Experiments

Evaluation

<table>
<thead>
<tr>
<th>Bound</th>
<th>KoAT</th>
<th>Absynth</th>
<th>eco-imp</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(1)$</td>
<td>8</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>$O(n)$</td>
<td>42</td>
<td>35</td>
<td>35</td>
</tr>
<tr>
<td>$O(n^2)$</td>
<td>15</td>
<td>9</td>
<td>15</td>
</tr>
<tr>
<td>$O(n^{2^2})$</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>EXP</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$\infty$</td>
<td>7</td>
<td>15</td>
<td>14</td>
</tr>
<tr>
<td>TO</td>
<td>0</td>
<td>9</td>
<td>3</td>
</tr>
</tbody>
</table>

- Successful runs: 91% KoAT, 68% Absynth, 77% eco-imp.
Experiments

Evaluation

<table>
<thead>
<tr>
<th>Bound</th>
<th>KoAT</th>
<th>Absynth</th>
<th>eco-imp</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(1)$</td>
<td>8</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>$O(n)$</td>
<td>42</td>
<td>35</td>
<td>35</td>
</tr>
<tr>
<td>$O(n^2)$</td>
<td>15</td>
<td>9</td>
<td>15</td>
</tr>
<tr>
<td>$O(n^{2^2})$</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>EXP</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$\infty$</td>
<td>7</td>
<td>15</td>
<td>14</td>
</tr>
<tr>
<td>TO</td>
<td>0</td>
<td>9</td>
<td>3</td>
</tr>
</tbody>
</table>

- Successful runs: 91% KoAT, 68% Absynth, 77% eco-imp.
- KoAT especially strong on large examples with many loops but only few randomization.
Conclusion

Summary

• Presented novel modular approach for inferring upper bounds on expected runtimes.
  • Core idea: alternate computation of bounds on expected runtimes and expected sizes.
  • Approach showed very good results in experimental evaluation.

Future Work

• Switch order of analysis (top-down → bottom-up).
• Generalize approach to cost-bounds.
• Combination with underlying approach of Absynth resp. eco-imp.
Conclusion

Summary

• Presented novel modular approach for inferring upper bounds on expected runtimes.
Conclusion

Summary

• Presented novel modular approach for inferring upper bounds on expected runtimes.
• Core idea: alternate computation of bounds on expected runtimes and expected sizes.

Future Work

• Switch order of analysis (top-down $\rightarrow$ bottom-up).
• Generalize approach to cost-bounds.
• Combination with underlying approach of Absynth resp. eco-imp.
Conclusion

Summary

- Presented novel modular approach for inferring upper bounds on expected runtimes.
- Core idea: alternate computation of bounds on expected runtimes and expected sizes.
- Approach showed very good results in experimental evaluation.
Conclusion

Summary

• Presented novel modular approach for inferring upper bounds on expected runtimes.
• Core idea: alternate computation of bounds on expected runtimes and expected sizes.
• Approach showed very good results in experimental evaluation.
Conclusion

Summary

- Presented novel modular approach for inferring upper bounds on expected runtimes.
- Core idea: alternate computation of bounds on expected runtimes and expected sizes.
- Approach showed very good results in experimental evaluation.

Future Work
Conclusion

Summary

• Presented novel modular approach for inferring upper bounds on expected runtimes.
• Core idea: alternate computation of bounds on expected runtimes and expected sizes.
• Approach showed very good results in experimental evaluation.

Future Work

• Switch order of analysis (top-down → bottom-up).
Conclusion

Summary

• Presented novel modular approach for inferring upper bounds on expected runtimes.
• Core idea: alternate computation of bounds on expected runtimes and expected sizes.
• Approach showed very good results in experimental evaluation.

Future Work

• Switch order of analysis (top-down $\rightarrow$ bottom-up).
• Generalize approach to cost-bounds.
Conclusion

Summary

• Presented novel modular approach for inferring upper bounds on expected runtimes.
• Core idea: alternate computation of bounds on expected runtimes and expected sizes.
• Approach showed very good results in experimental evaluation.

Future Work

• Switch order of analysis (top-down → bottom-up).
• Generalize approach to cost-bounds.
• Combination with underlying approach of Absynth resp. eco-imp.
Conclusion

Summary

- Presented novel modular approach for inferring upper bounds on expected runtimes.
- Core idea: alternate computation of bounds on expected runtimes and expected sizes.
- Approach showed very good results in experimental evaluation.

Future Work

- Switch order of analysis (top-down $\rightarrow$ bottom-up).
- Generalize approach to cost-bounds.
- Combination with underlying approach of Absynth resp. eco-imp.
Conclusion

Summary

• Presented novel modular approach for inferring upper bounds on expected runtimes.
• Core idea: alternate computation of bounds on expected runtimes and expected sizes.
• Approach showed very good results in experimental evaluation.

Future Work

• Switch order of analysis (top-down → bottom-up).
• Generalize approach to cost-bounds.
• Combination with underlying approach of Absynth resp. eco-imp.

Thank you