Guaranteed Bounds on Posterior Distributions of Discrete Probabilistic Programs with Loops

Fabian Zaiser[1] Andrzej Murawski[1] Luke Ong[1,2]

[1]University of Oxford [2]Nanyang Technological University

POPL, 2025-01-23

A probabilistic puzzle

- *You throw a fair six-sided die repeatedly until you get a 6.*
- *You observe only even numbers during the throws.*
- *What is the expected number of throws (including the 6) conditioned on this event?*

Probabilistic Programming

$Throws := 0;$

$Die := 0;$

$\textsf{while}\ Die \neq 6\ \{$

$\quad Die \sim \textsf{Uniform}\{1, \ldots, 6\};$

$\quad \textsf{observe}\ Die \in \{2, 4, 6\};$

$\quad Throws \mathrel{+}= 1$

$\}$

Query: $\mathbb{E}[Throws]$

Probabilistic Programming

$Throws := 0;$

$Die := 0;$

$\textsf{while}\ Die \neq 6\ \{$

$\quad Die \sim \textsf{Uniform}\{1, \ldots, 6\};$

$\quad \textsf{observe}\ Die \in \{2, 4, 6\};$

$\quad Throws \mathrel{+}= 1$

$\}$

Query: $\mathbb{E}[Throws]$

Challenges

- infinite support
- observations & conditioning
- unbounded loops

Probabilistic Programming

$Throws := 0;$

$Die := 0;$

while $Die \neq 6$ {

  $Die \sim$ Uniform$\{1, \ldots, 6\}$;

  observe $Die \in \{2, 4, 6\}$;

  $Throws \mathrel{+}= 1$

}

Query: $\mathbb{E}[Throws]$

Challenges

- infinite support
- observations & conditioning
- unbounded loops

No existing tool for rigorous & automatic analysis!

Exact techniques

- ✓ precise result
- ✗ often intractable
- ✗ or require user annotations

Approximate methods

- ✓ always applicable
- ✗ no guarantees

Exact techniques

✓ precise result

✗ often intractable

✗ or require user annotations

Guaranteed bounds

✓ often applicable

✓ hard guarantees: $\mathbb{P}[X = a] \in [l, u]$

Approximate methods

✓ always applicable

✗ no guarantees

Exact techniques

✓ precise result

✗ often intractable

✗ or require user annotations

Guaranteed bounds

✓ often applicable

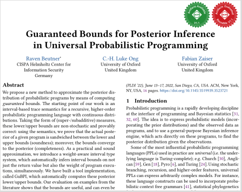
✓ hard guarantees:

$$\mathbb{P}[X = a] \in [l, u]$$

Approximate methods

✓ always applicable

✗ no guarantees



Guaranteed Bounds for Posterior Inference in Universal Probabilistic Programming

Raven Beutner*
CISPA Helmholtz Center for Information Security
Germany

C.-H. Luke Ong
University of Oxford
United Kingdom

Fabian Zaiser
University of Oxford
United Kingdom

Abstract

We propose a new method to approximate the posterior distribution of probabilistic programs by means of computing *guaranteed* bounds. The starting point of our work is an interval-based trace semantics for a recursive, higher-order probabilistic programming language with continuous distributions. Taking the form of (super-/subadditive) measures, these lower/upper bounds are non-stochastic and provably correct: using the semantics, we prove that the actual posterior of a given program is sandwiched between the lower and upper bounds (soundness); moreover, the bounds converge to the posterior (completeness). As a practical and sound approximation, we introduce a weight-aware interval type system, which automatically infers interval bounds on not just the return value but also the weight of program executions, simultaneously. We have built a tool implementation, called GuBPI, which automatically computes these posterior lower/upper bounds. Our evaluation on examples from the literature shows that the bounds are useful, and can even be

(PLDI '22), June 13–17, 2022, San Diego, CA, USA. ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3519939.3523721

1 Introduction

Probabilistic programming is a rapidly developing discipline at the interface of programming and Bayesian statistics [31, 32, 60]. The idea is to express probabilistic models (incorporating the prior distributions) and the observed data as programs, and to use a general-purpose Bayesian inference engine, which acts directly on these programs, to find the posterior distribution given the observations.

Some of the most influential probabilistic programming languages (PPLs) used in practice are *universal* (i.e. the underlying language is Turing-complete); e.g. Church [30], Anglican [59], Gen [18], Pyro [6], and Turing [24]. Using stochastic branching, recursion, and higher-order features, universal PPLs can express arbitrarily complex models. For instance, these language constructs can be used to incorporate probabilistic context free grammars [41], statistical phylogenetics

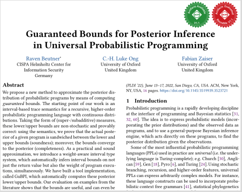
[Beutner et al., PLDI 2022]

Why guaranteed bounds?

→ safety properties (quantitative program verification)

→ ground truth to debug approximate methods

Exact techniques	Guaranteed bounds	Approximate methods
✓ precise result	✓ often applicable	✓ always applicable
✗ often intractable	✓ hard guarantees: $\mathbb{P}[X = a] \in [l, u]$	✗ no guarantees
✗ or require user annotations		



Guaranteed Bounds for Posterior Inference in Universal Probabilistic Programming

Raven Beutner* — CISPA Helmholtz Center for Information Security, Germany
C.-H. Luke Ong — University of Oxford, United Kingdom
Fabian Zaiser — University of Oxford, United Kingdom

Abstract

We propose a new method to approximate the posterior distribution of probabilistic programs by means of computing *guaranteed* bounds. The starting point of our work is an interval-based trace semantics for a recursive, higher-order probabilistic programming language with continuous distributions. Taking the form of (super-/subadditive) measures, these lower/upper bounds are non-stochastic and provably correct: using the semantics, we prove that the actual posterior of a given program is sandwiched between the lower and upper bounds (soundness); moreover, the bounds converge to the posterior (completeness). As a practical and sound approximation, we introduce a weight-aware interval type system, which automatically infers interval bounds on not just the return value but also the weight of program executions, simultaneously. We have built a tool implementation, called GuBPI, which automatically computes these posterior lower/upper bounds. Our evaluation on examples from the literature shows that the bounds are useful, and can even be

(PLDI '22), June 13–17, 2022, San Diego, CA, USA. ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3519939.3523721

1 Introduction

Probabilistic programming is a rapidly developing discipline at the interface of programming and Bayesian statistics [31, 32, 60]. The idea is to express probabilistic models (incorporating the prior distributions) and the observed data as programs, and to use a general-purpose Bayesian inference engine, which acts directly on these programs, to find the posterior distribution given the observations.

Some of the most influential probabilistic programming languages (PPLs) used in practice are *universal* (i.e. the underlying language is Turing-complete); e.g. Church [30], Anglican [59], Gen [18], Pyro [6], and Turing [24]. Using stochastic branching, recursion, and higher-order features, universal PPLs can express arbitrarily complex models. For instance, these language constructs can be used to incorporate probabilistic context free grammars [41], statistical phylogenetics

[Beutner et al., PLDI 2022]

Why guaranteed bounds?

→ safety properties (quantitative program verification)

→ ground truth to debug approximate methods

Previous work on guaranteed bounds

✗ has unnecessary overhead for discrete programs

✗ cannot bound moments and tails

Problem Statement

For a discrete probabilistic program with variables in $\mathbb{N}$, with conditioning, and with unbounded loops,

Problem Statement

For a discrete probabilistic program with variables in $\mathbb{N}$, with conditioning, and with unbounded loops, we want to automatically find bounds on

- its probabilities: $\mathbb{P}[X = n] \in [l, u]$,
- its k-th moments: $\mathbb{E}[X^k] \in [l, u]$,
- its tail asymptotics: $\mathbb{P}[X = n] \leq O(c^n)$ for $c < 1$.

Problem Statement

For a discrete probabilistic program with variables in $\mathbb{N}$, with conditioning, and with unbounded loops, we want to automatically find bounds on

- its probabilities: $\mathbb{P}[X = n] \in [l, u]$,
- its k-th moments: $\mathbb{E}[X^k] \in [l, u]$,
- its tail asymptotics: $\mathbb{P}[X = n] \leq O(c^n)$ for $c < 1$.

Two approaches

- Residual mass semantics
- Geometric bound semantics

Programming Language

Imperative language with discrete variables $X_1, \ldots, X_n$ taking values in $\mathbb{N}$.

$$\text{Programs } P ::= \text{skip} \mid P_1; P_2 \mid X_k \mathrel{+}= a \mid X_k \mathrel{\dot{-}}= 1 \mid X_k \sim \text{Bernoulli}(\rho)$$
$$\mid \text{if } E \,\{P_1\}\, \text{else} \,\{P_2\} \mid \text{while } E \,\{P\} \mid \text{observe } E$$
$$\text{Events } E ::= X_k = a \mid \neg E \mid E_1 \wedge E_2$$
$$\text{where } \rho \in [0,1], \quad a \in \mathbb{N}$$

Programming Language

Imperative language with discrete variables $X_1, \ldots, X_n$ taking values in $\mathbb{N}$.

$$\text{Programs } P ::= \text{skip} \mid P_1; P_2 \mid X_k \mathrel{+}= a \mid X_k \mathrel{\dot{-}}= 1 \mid X_k \sim \text{Bernoulli}(\rho)$$
$$\mid \text{if } E \, \{P_1\} \text{ else } \{P_2\} \mid \text{while } E \, \{P\} \mid \text{observe } E$$
$$\text{Events } E ::= X_k = a \mid \neg E \mid E_1 \wedge E_2$$
$$\text{where } \rho \in [0,1], \quad a \in \mathbb{N}$$

Expressivity

- Turing complete
- Geometric & negative binomial distributions + all finite discrete distributions
- some constructs difficult to encode, e.g. Poisson distribution

Semantics

$[\![P]\!]$ transforms distributions on the state space $\mathbb{N}^n$:

$$\boxed{\text{Initial distribution } \mu} \xrightarrow{\text{Event } E} \boxed{\text{“restriction” } \mu|_E}$$

$$\boxed{\text{Initial distribution } \mu} \xrightarrow{\text{Statement } P} \boxed{\text{result distribution } [\![P]\!](\mu)}$$

Semantics

$[\![P]\!]$ transforms distributions on the state space $\mathbb{N}^n$:

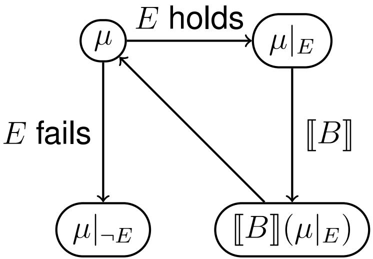
$$\text{Initial distribution } \mu \xrightarrow{\text{Event } E} \text{“restriction” } \mu|_E$$

$$\text{Initial distribution } \mu \xrightarrow{\text{Statement } P} \text{result distribution } [\![P]\!](\mu)$$

- distribution at the start of the program: Dirac$(0, \ldots, 0)$
- ignore normalization in this talk

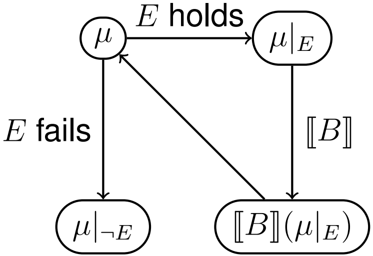
Semantics of loops

while $E\,\{B\}$



Semantics of loops

$\text{while}\ E\ \{B\}$

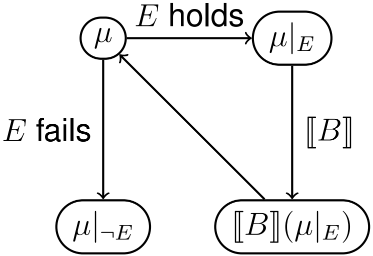


Fixpoint equation

$$[\![\text{while}\ E\ \{B\}]\!](\mu) =$$

Semantics of loops

while $E\,\{B\}$

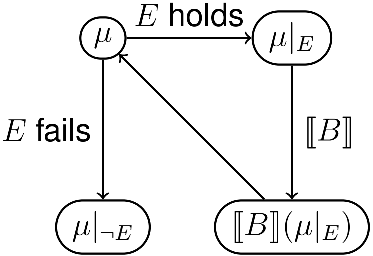


Fixpoint equation

$$[\![\text{while } E\,\{B\}]\!](\mu) = \underbrace{\mu|_{\neg E}}_{\text{loop exit}}$$

Semantics of loops

while $E\ \{B\}$



Fixpoint equation

$$[\![\text{while } E\ \{B\}]\!](\mu) = \underbrace{\mu|_{\neg E}}_{\text{loop exit}} + [\![\text{while } E\ \{B\}]\!]\underbrace{([\![B]\!](\mu|_E))}_{\text{one iteration}}$$

Lower bounds — the easy part

Lower bounds — the easy part

Unroll the loop a few times (Kleene iteration):

Lower bounds — the easy part

Unroll the loop a few times (Kleene iteration):

$$\boxed{[\![\textsf{while}\, E\,\{P\}]\!]}(\mu) = \mu|_{\neg E} + \boxed{[\![\textsf{while}\, E\,\{P\}]\!]}([\![P]\!](\mu|_E))$$

Lower bounds — the easy part

Unroll the loop a few times (Kleene iteration):

$$\begin{aligned}\boxed{[\![\textsf{while}\, E\,\{P\}]\!]}(\mu) &= \mu|_{\neg E} + \boxed{[\![\textsf{while}\, E\,\{P\}]\!]}([\![P]\!](\mu|_E)) \\ &= \mu|_{\neg E} + [\![P]\!](\mu|_E)|_{\neg E} + \underbrace{\boxed{[\![\textsf{while}\, E\,\{P\}]\!]}([\![P]\!]([\![P]\!](\mu|E)|_E))}_{\succeq \mathbf{0}}\end{aligned}$$

Lower bounds — the easy part

Unroll the loop a few times (Kleene iteration):

$$
\begin{aligned}
\llbracket \text{while}\, E\, \{P\} \rrbracket(\mu) &= \mu|_{\neg E} + \llbracket \text{while}\, E\, \{P\} \rrbracket(\llbracket P \rrbracket(\mu|_E)) \\
&= \mu|_{\neg E} + \llbracket P \rrbracket(\mu|_E)|_{\neg E} + \underbrace{\llbracket \text{while}\, E\, \{P\} \rrbracket(\llbracket P \rrbracket(\llbracket P \rrbracket(\mu|E)|_E))}_{\succeq \mathbf{0}} \\
&\succeq \mu|_{\neg E} + \llbracket P \rrbracket(\mu|_E)|_{\neg E}
\end{aligned}
$$

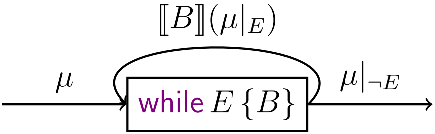
Lower bounds — the easy part

Unroll the loop a few times (Kleene iteration):

$$\begin{aligned}
\boxed{[\![\text{while}\, E\, \{P\}]\!]}(\mu) &= \mu|_{\neg E} + \boxed{[\![\text{while}\, E\, \{P\}]\!]}([\![P]\!](\mu|_E)) \\
&= \mu|_{\neg E} + [\![P]\!](\mu|_E)|_{\neg E} + \underbrace{\boxed{[\![\text{while}\, E\, \{P\}]\!]}([\![P]\!]([\![P]\!](\mu|E)|_E))}_{\succeq \mathbf{0}} \\
&\succeq \mu|_{\neg E} + [\![P]\!](\mu|_E)|_{\neg E}
\end{aligned}$$

- easy to compute: only finite discrete distributions involved
- converges to true distribution with increasing unrolling

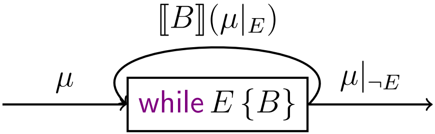
Upper bounds: residual mass



Flow of total probability mass $\mu(\mathbb{N}^n)$

- initially 1
- in every iteration, some mass "flows" out of the loop
- can bound the **residual mass** after unrolling

Upper bounds: residual mass

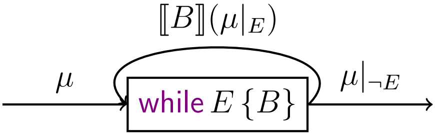


Flow of total probability mass $\mu(\mathbb{N}^n)$

- initially 1
- in every iteration, some mass "flows" out of the loop
- can bound the **residual mass** after unrolling

$$\underbrace{[\![P]\!]_{\text{res}}(\mu)}_{\text{residual mass}} = \underbrace{\mu(\mathbb{N}^n)}_{\text{initial mass}} - \underbrace{[\![P]\!]_{\text{lo}}(\mu)(\mathbb{N}^n)}_{\text{lower bound on mass}}$$

Upper bounds: residual mass



Flow of total probability mass $\mu(\mathbb{N}^n)$

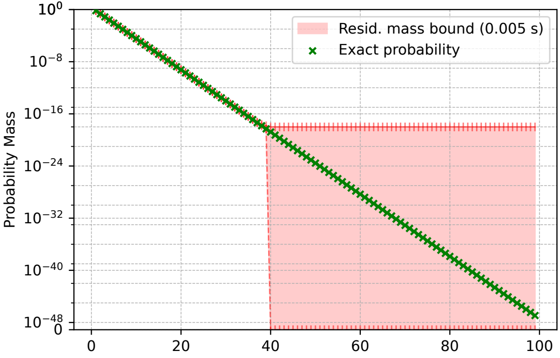
- initially 1
- in every iteration, some mass "flows" out of the loop
- can bound the **residual mass** after unrolling

$$\underbrace{[\![P]\!]_{\mathsf{res}}(\mu)}_{\text{residual mass}} = \underbrace{\mu(\mathbb{N}^n)}_{\text{initial mass}} - \underbrace{[\![P]\!]_{\mathsf{lo}}(\mu)(\mathbb{N}^n)}_{\text{lower bound on mass}}$$

The probability of S at the end of the program P is bounded by:

$$\underbrace{[\![P]\!](\mu)(S)}_{\text{probability of } S} \preceq \underbrace{[\![P]\!]_{\mathsf{lo}}(\mu)(S)}_{\text{lower bound}} + \underbrace{[\![P]\!]_{\mathsf{res}}(\mu)}_{\text{residual mass}}$$

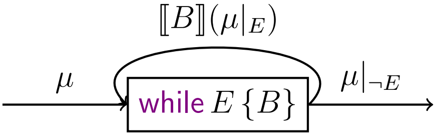
Residual mass: in practice



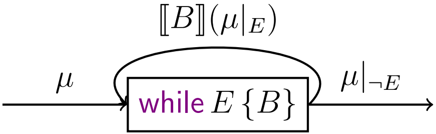
- ✓ bounds on probability masses
- ✓ speedup compared to previous work: $100\times$ to $10^5\times$
- ✗ flat tail bounds
- ✗ cannot bound moments

$\rightarrow$ need more informative bounds

Upper bounds — part 2

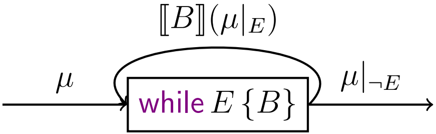


Upper bounds — part 2



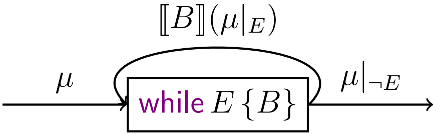
What if $[\![B]\!](\mu|_E) \preceq \mu$?

Upper bounds — part 2



What if $[\![B]\!](\mu|_E) \preceq c \cdot \mu$ for $c < 1$?

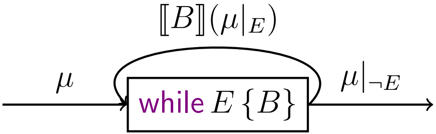
Upper bounds — part 2



What if $[\![B]\!](\mu|_E) \preceq c \cdot \mu$ for $c < 1$?

$$[\![\text{while } E\ \{B\}]\!](\mu) = \mu|_{\neg E} + [\![\text{while } E\ \{B\}]\!]\big(\underbrace{[\![B]\!](\mu|_E)}_{\preceq c\cdot\mu}\big)$$

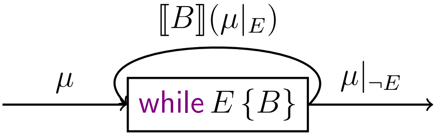
Upper bounds — part 2



What if $[\![B]\!](\mu|_E) \preceq c \cdot \mu$ for $c < 1$?

$$\begin{aligned}[\![\text{while } E\ \{B\}]\!](\mu) &= \mu|_{\neg E} + [\![\text{while } E\ \{B\}]\!](\underbrace{[\![B]\!](\mu|_E)}_{\preceq c\cdot\mu}) \\ &\preceq \mu|_{\neg E} + c \cdot [\![\text{while } E\ \{B\}]\!](\mu)\end{aligned}$$

Upper bounds — part 2



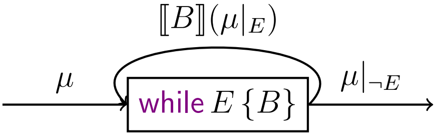
What if $[\![B]\!](\mu|_E) \preceq c \cdot \mu$ for $c < 1$?

$$[\![\text{while } E\ \{B\}]\!](\mu) = \mu|_{\neg E} + [\![\text{while } E\ \{B\}]\!]\big(\underbrace{[\![B]\!](\mu|_E)}_{\preceq c\cdot\mu}\big)$$

$$\preceq \mu|_{\neg E} + c \cdot [\![\text{while } E\ \{B\}]\!](\mu)$$

$$\implies (1-c)\cdot [\![\text{while } E\ \{B\}]\!](\mu) \preceq \mu|_{\neg E}$$

Upper bounds — part 2



What if $[\![B]\!](\mu|_E) \preceq c \cdot \mu$ for $c < 1$?

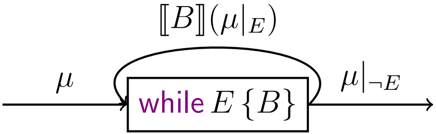
$$[\![\text{while } E\ \{B\}]\!](\mu) = \mu|_{\neg E} + [\![\text{while } E\ \{B\}]\!](\underbrace{[\![B]\!](\mu|_E)}_{\preceq c\cdot\mu})$$

$$\preceq \mu|_{\neg E} + c \cdot [\![\text{while } E\ \{B\}]\!](\mu)$$

$$\implies (1-c)\cdot [\![\text{while } E\ \{B\}]\!](\mu) \preceq \mu|_{\neg E}$$

$$\implies [\![\text{while } E\ \{B\}]\!](\mu) \preceq \frac{1}{1-c} \cdot \mu|_{\neg E}$$

Upper bounds — part 2



What if $[\![B]\!](\mu|_E) \preceq c \cdot \mu$ for $c < 1$?

$$[\![\text{while } E\,\{B\}]\!](\mu) = \mu|_{\neg E} + [\![\text{while } E\,\{B\}]\!](\underbrace{[\![B]\!](\mu|_E)}_{\preceq c \cdot \mu})$$

$$\preceq \mu|_{\neg E} + c \cdot [\![\text{while } E\,\{B\}]\!](\mu)$$

$$\implies (1-c) \cdot [\![\text{while } E\,\{B\}]\!](\mu) \preceq \mu|_{\neg E}$$

$$\implies [\![\text{while } E\,\{B\}]\!](\mu) \preceq \frac{1}{1-c} \cdot \mu|_{\neg E}$$

✗ The initial distribution μ rarely decreases uniformly by a factor of $c < 1$.

→ Find $\nu \succeq \mu$ satisfying the condition! (“Strengthen the induction hypothesis”)

Contraction invariant

- Let $P = \text{while}\, E\, \{B\}$ be a loop.
- Let μ be an initial distribution on $\mathbb{N}^n$.
- A **contraction invariant** is a distribution ν such that

$$\mu \preceq \nu \qquad \text{and} \qquad [\![B]\!](\nu|_C) \preceq c \cdot \nu \text{ where } c < 1$$

Contraction invariant

- Let $P = \textsf{while}\, E\,\{B\}$ be a loop.
- Let μ be an initial distribution on $\mathbb{N}^n$.
- A **contraction invariant** is a distribution ν such that

$$\mu \preceq \nu \qquad \text{and} \qquad [\![B]\!](\nu|_C) \preceq c \cdot \nu \text{ where } c < 1$$

If ν is a contraction invariant for $\textsf{while}\, E\,\{B\}$ and μ then

$$[\![\textsf{while}\, E\,\{B\}]\!](\mu) \preceq \frac{1}{1-c} \cdot \nu|_{\neg E}$$

Contraction invariant

- Let $P = \text{while}\, E\, \{B\}$ be a loop.
- Let μ be an initial distribution on $\mathbb{N}^n$.
- A **contraction invariant** is a distribution ν such that

$$\mu \preceq \nu \qquad \text{and} \qquad [\![B]\!](\nu|_C) \preceq c \cdot \nu \text{ where } c < 1$$

If ν is a contraction invariant for $\text{while}\, E\, \{B\}$ and μ then

$$[\![\text{while}\, E\, \{B\}]\!](\mu) \preceq \frac{1}{1-c} \cdot \nu|_{\neg E}$$

How do we find a contraction invariant?

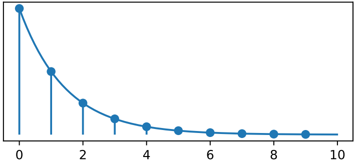
Candidates for Contraction Invariants

We need to reason about tails!

Candidates for Contraction Invariants

We need to reason about tails!

$$p(k) = \rho \cdot (1-\rho)^k$$

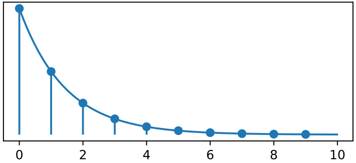


Geometric distribution?

Candidates for Contraction Invariants

We need to reason about tails!

$$p(k) = \rho \cdot (1 - \rho)^k$$



Geometric distribution?

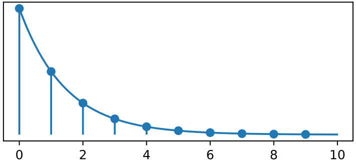
✓ moments, tails are easy

✗ not closed under many program operations (e.g. increment)

Candidates for Contraction Invariants

We need to reason about tails!

$$p(k) = \rho \cdot (1-\rho)^k$$



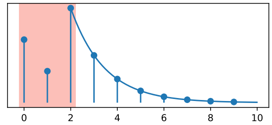
Geometric distribution?

✓ moments, tails are easy

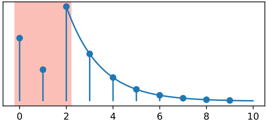
✗ not closed under many program operations (e.g. increment)

$\to$ Generalize!

Eventually Geometric Distributions (EGDs)



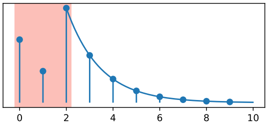
Eventually Geometric Distributions (EGDs)



$$\text{EGD}\Big(\underbrace{\big(p_0 \quad p_1 \quad p_2\big)}_{\text{initial block}}, \underbrace{\alpha}_{\text{decay rate}}\Big)$$

k	0	1	2	3	4	$\ldots$
$p(k)$	p_0	p_1	p_2	$p_2\alpha$	$p_2\alpha^2$	$\ldots$

Eventually Geometric Distributions (EGDs)



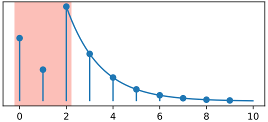
$$\mathrm{EGD}\Big(\underbrace{\begin{pmatrix} p_0 & p_1 & p_2 \end{pmatrix}}_{\text{initial block}},\ \underbrace{\alpha}_{\text{decay rate}}\Big)$$

k	0	1	2	3	4	…
$p(k)$	p_0	p_1	p_2	$p_2\alpha$	$p_2\alpha^2$	…

$$\mathrm{EGD}\left(\begin{pmatrix} q_{0,0} & q_{0,1} \\ q_{1,0} & q_{1,1} \end{pmatrix}, (\alpha, \beta)\right)$$

	0	1	2	3	…
0	$q_{0,0}$	$q_{0,1}$	$q_{0,1} \cdot \alpha$	$q_{0,1} \cdot \alpha^2$	…
1	$q_{1,0}$	$q_{1,1}$	$q_{1,1} \cdot \alpha$	$q_{1,1} \cdot \alpha^2$	…
2	$q_{1,0} \cdot \beta$	$q_{1,1} \cdot \beta$	$q_{1,1} \cdot \alpha \cdot \beta$	$q_{1,1} \cdot \alpha^2 \cdot \beta$	…
3	$q_{1,0} \cdot \beta^2$	$q_{1,1} \cdot \beta^2$	$q_{1,1} \cdot \alpha \cdot \beta^2$	$q_{1,1} \cdot \alpha^2 \cdot \beta^2$	…
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Eventually Geometric Distributions (EGDs)



$$\text{EGD}\Big(\underbrace{\begin{pmatrix} p_0 & p_1 & p_2 \end{pmatrix}}_{\text{initial block}}, \underbrace{\alpha}_{\text{decay rate}}\Big)$$

k	0	1	2	3	4	…
$p(k)$	p_0	p_1	p_2	$p_2\alpha$	$p_2\alpha^2$	…

$$\text{EGD}\left(\begin{pmatrix} q_{0,0} & q_{0,1} \\ q_{1,0} & q_{1,1} \end{pmatrix}, (\alpha, \beta)\right)$$

	0	1	2	3	…
0	$q_{0,0}$	$q_{0,1}$	$q_{0,1} \cdot \alpha$	$q_{0,1} \cdot \alpha^2$	…
1	$q_{1,0}$	$q_{1,1}$	$q_{1,1} \cdot \alpha$	$q_{1,1} \cdot \alpha^2$	…
2	$q_{1,0} \cdot \beta$	$q_{1,1} \cdot \beta$	$q_{1,1} \cdot \alpha \cdot \beta$	$q_{1,1} \cdot \alpha^2 \cdot \beta$	…
3	$q_{1,0} \cdot \beta^2$	$q_{1,1} \cdot \beta^2$	$q_{1,1} \cdot \alpha \cdot \beta^2$	$q_{1,1} \cdot \alpha^2 \cdot \beta^2$	…
⋮	⋮	⋮	⋮	⋮	⋱

✓ easy to compute probababilities, moments, tail asymptotics

✓ closed under many operations

Semantics for EGDs

- need a semantics that operates on EGDs & yields upper bounds
- not a function: there may be many valid upper bounds

Semantics for EGDs

- need a semantics that operates on EGDs & yields upper bounds
- not a function: there may be many valid upper bounds

Geometric bound semantics

$[\![P]\!]^{\text{geo}}$ is a **relational** semantics on EGDs

$$(\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha}), \mathsf{EGD}(\mathbf{Q}, \boldsymbol{\beta})) \in [\![P]\!]^{\text{geo}}$$

Semantics for EGDs

- need a semantics that operates on EGDs & yields upper bounds
- not a function: there may be many valid upper bounds

Geometric bound semantics

$[\![P]\!]^{\text{geo}}$ is a **relational** semantics on EGDs

$$(\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha}), \mathsf{EGD}(\mathbf{Q}, \boldsymbol{\beta})) \in [\![P]\!]^{\text{geo}}$$

- ensures the **bound**: $[\![P]\!](\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha})) \preceq \mathsf{EGD}(\mathbf{Q}, \boldsymbol{\beta})$

Semantics for EGDs

- need a semantics that operates on EGDs & yields upper bounds
- not a function: there may be many valid upper bounds

Geometric bound semantics

$[\![P]\!]^{\text{geo}}$ is a **relational** semantics on EGDs

$$(\text{EGD}(\mathbf{P}, \boldsymbol{\alpha}), \text{EGD}(\mathbf{Q}, \boldsymbol{\beta})) \in [\![P]\!]^{\text{geo}}$$

- ensures the **bound**: $[\![P]\!](\text{EGD}(\mathbf{P}, \boldsymbol{\alpha})) \preceq \text{EGD}(\mathbf{Q}, \boldsymbol{\beta})$
- reduces to **polynomial inequalities** in the parameters $\mathbf{P}, \mathbf{Q}, \boldsymbol{\alpha}, \boldsymbol{\beta}$

Semantics for EGDs

- need a semantics that operates on EGDs & yields upper bounds
- not a function: there may be many valid upper bounds

Geometric bound semantics

$[\![P]\!]^{\text{geo}}$ is a **relational** semantics on EGDs

$$(\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha}), \mathsf{EGD}(\mathbf{Q}, \boldsymbol{\beta})) \in [\![P]\!]^{\text{geo}}$$

- ensures the **bound**: $[\![P]\!](\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha})) \preceq \mathsf{EGD}(\mathbf{Q}, \boldsymbol{\beta})$
- reduces to **polynomial inequalities** in the parameters $\mathbf{P}, \mathbf{Q}, \boldsymbol{\alpha}, \boldsymbol{\beta}$
- can **decide** the existence of an upper bound $\mathsf{EGD}(\mathbf{Q}, \boldsymbol{\beta})$!

Theoretical results

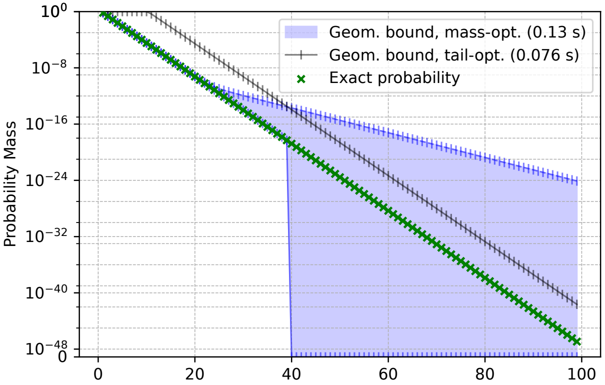
Soundness: Residual mass semantics and geometric bound semantics are sound.

Convergence: The bounds for both semantics converge in total variation distance, as loops are unrolled further and further.*

Existence: We proved some sufficient and some necessary conditions for the existence of geometric bounds.

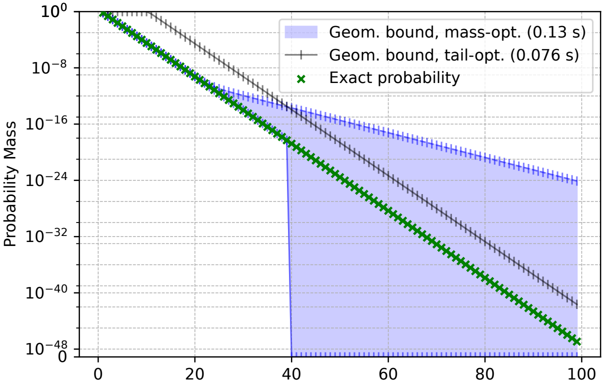
Experimental results

$$
\begin{array}{l}
Throws := 0; \\
Die := 0; \\
\textsf{while } Die \neq 6 \, \{ \\
\quad Die \sim \textsf{Uniform}\{1, \ldots, 6\}; \\
\quad \textsf{observe } Die \in \{2, 4, 6\}; \\
\quad Throws \mathrel{+}= 1 \\
\}
\end{array}
$$



Experimental results

$$
\begin{array}{l}
\mathit{Throws} := 0; \\
\mathit{Die} := 0; \\
\textsf{while } \mathit{Die} \neq 6 \, \{ \\
\quad \mathit{Die} \sim \textsf{Uniform}\{1, \ldots, 6\}; \\
\quad \textsf{observe } \mathit{Die} \in \{2, 4, 6\}; \\
\quad \mathit{Throws} \mathrel{+}= 1 \\
\}
\end{array}
$$



	$\mathbb{E}[\mathit{Throws}]$	tail
exact	1.5	$\Theta((1/3)^n)$
bounds	$[1.4999999999999998, 1.5000008]$	$O(0.34^n)$

Experimental results

Applicability

- collected 43 benchmarks from literature
- finds bounds for 37 (85%) of benchmarks
- many could not be automatically analyzed before

Experimental results

Applicability

- collected 43 benchmarks from literature
- finds bounds for 37 (85%) of benchmarks
- many could not be automatically analyzed before

Performance

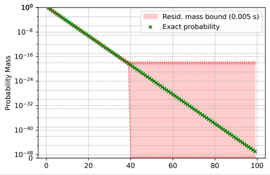
- running time: usually a few seconds, up to 5 minutes
- quality of bounds: usually very tight; worse for heavy-tailed distributions
- comparison with previous tools: supports more benchmarks, often faster

Guaranteed Bounds on Posterior Distributions of Discrete Probabilistic Programs with Loops

Lower bounds: unrolling & cutting off loops

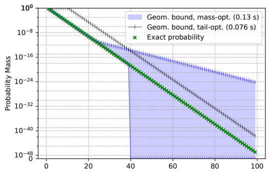
Residual mass semantics: flat bound on residual distribution missed by the lower bound

- faster than previous methods
- bounds on probabilities



Geometric bound semantics: upper bounds with geometric tails

- operates on EGDs (eventually geometric distributions)
- contraction invariants: distribution decreases by factor $c < 1$ each iteration
- reduces to polynomial inequality constraints
- can bound probabilities, moments, tails



Backup slides

Implementation

Solving polynomial constraints

- existential theory of the reals is decidable
- SMT solvers are usually too slow
- IPOPT, numerical solver, works well
- numerical results are verified with exact arithmetic

Implementation

Solving polynomial constraints

- existential theory of the reals is decidable
- SMT solvers are usually too slow
- IPOPT, numerical solver, works well
- numerical results are verified with exact arithmetic

Optimizing the bound

- want bounds that minimize some objective: expected value / tail decay rate / …
- use numerical optimization

Limitations

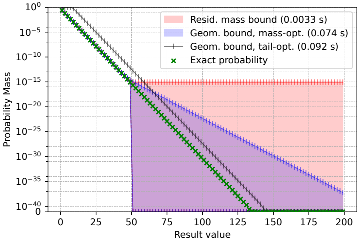
Programming language:

- no negative or continuous variables
- some distributions (e.g. Poisson) are difficult to encode
- no symbolic inputs

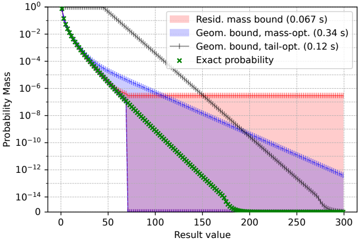
Geometric bound semantics:

- incompleteness: bounds may not exist
- solving polynomial constraints may be too difficult
- tail bounds do not converge
- correlations between variables cannot be represented

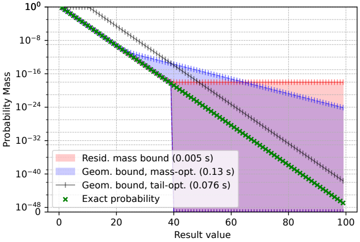
More plots



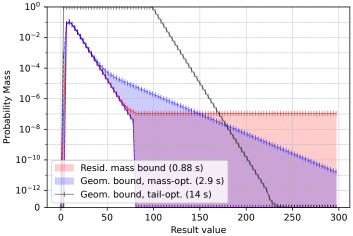
(a) Geom. counter



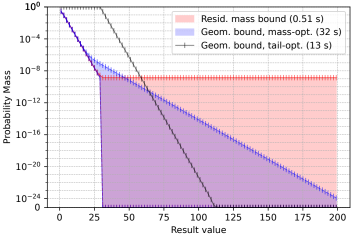
(b) Asym. rand. walk



(c) Die paradox



(d) Coupon collector problem with 5 coupons



(e) Herman's self-stabilization with 3 processes