Guaranteed Bounds on Posterior Distributions of Discrete Probabilistic Programs with Loops

FABIAN ZAISER, University of Oxford, United Kingdom
ANDRZEJ S. MURAWSKI, University of Oxford, United Kingdom
C.-H. LUKE ONG, Nanyang Technological University, Singapore

We study the problem of bounding the posterior distribution of discrete probabilistic programs with unbounded support, loops, and conditioning. Loops pose the main difficulty in this setting: even if exact Bayesian inference is possible, the state of the art requires user-provided loop invariant templates. By contrast, we aim to find *guaranteed bounds*, which sandwich the true distribution. They are fully automated, applicable to more programs and provide more provable guarantees than approximate sampling-based inference. Since lower bounds can be obtained by unrolling loops, the main challenge is upper bounds, and we attack it in two ways. The first is called *residual mass semantics*, which is a flat bound based on the residual probability mass of a loop. The approach is simple, efficient, and has provable guarantees.

The main novelty of our work is the second approach, called *geometric bound semantics*. It operates on a novel family of distributions, called *eventually geometric distributions* (EGDs), and can bound the distribution of loops with a new form of loop invariants called *contraction invariants*. The invariant synthesis problem reduces to a system of polynomial inequality constraints, which is a decidable problem with automated solvers. If a solution exists, it yields an exponentially decreasing bound on the *whole* distribution, and can therefore bound moments and tail asymptotics as well, not just probabilities as in the first approach.

Both semantics enjoy desirable theoretical properties. In particular, we prove soundness and convergence, i.e. the bounds converge to the exact posterior as loops are unrolled further. We also investigate sufficient and necessary conditions for the existence of geometric bounds. On the practical side, we describe *Diabolo*, a fully-automated implementation of both semantics, and evaluate them on a variety of benchmarks from the literature, demonstrating their general applicability and the utility of the resulting bounds.

CCS Concepts: • **Mathematics of computing → Probabilistic inference problems**; • **Theory of computation → Program reasoning**; *Program semantics*; • **Software and its engineering → Formal methods**.

Additional Key Words and Phrases: probabilistic programming, Bayesian inference, program analysis, guaranteed bounds, posterior distribution, moments, tail asymptotics, loop invariant synthesis, quantitative analysis

ACM Reference Format:
Fabian Zaiser, Andrzej S. Murawski, and C.-H. Luke Ong. 2025. Guaranteed Bounds on Posterior Distributions of Discrete Probabilistic Programs with Loops. *Proc. ACM Program. Lang.* 9, POPL, Article 38 (January 2025), 32 pages. https://doi.org/10.1145/3704874

1 Introduction

Probabilistic programming is a discipline that studies programming languages with probabilistic constructs [Barthe et al. 2020]. The term is overloaded however. At the intersection with randomized algorithms and program analysis, it usually means a programming language with a construct for

Authors' Contact Information: Fabian Zaiser, University of Oxford, Oxford, United Kingdom, fabian.zaiser@cs.ox.ac.uk; Andrzej S. Murawski, University of Oxford, Oxford, United Kingdom, andrzej.murawski@cs.ox.ac.uk; C.-H. Luke Ong, Nanyang Technological University, Singapore, Singapore, luke.ong@ntu.edu.sg.

ACM 2475-1421/2025/1-ART38
https://doi.org/10.1145/3704874

probabilistic branching or sampling from probability distributions. As such, it is simply a language to express programs with random numbers and researchers study program analysis techniques for termination probabilities, safety properties, cost analysis, and others. At the intersection with statistics and machine learning, probabilistic programming is used to express (Bayesian) statistical models [van de Meent et al. 2018]. Bayesian inference is a very successful framework for reasoning and learning under uncertainty: it updates prior beliefs about the world with observed data to obtain posterior beliefs using Bayes' rule. As such, the programming languages for Bayesian models provide a construct for conditioning on data in addition to sampling from distributions. Since Bayesian inference is a difficult problem, a lot of research focuses on inference algorithms, in particular their correctness and efficiency. This paper contributes to both areas by developing methods to bound the distributions arising from probabilistic programs, especially those with loops.

Example 1.1. To illustrate the concept, consider the following puzzle due to Elchanan Mossel.

> 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?

This is a surprisingly tricky problem and most people get it wrong on the first try[1], based on the incorrect assumption that it is equivalent to throwing a die with only the three faces 2, 4, and 6. Probability theory and statistics abound with such counterintuitive results (e.g. the Monty-Hall problem), and probabilistic programming offers a precise way to disambiguate their description and make them amenable to automatic analysis and inference tools. Mossel's problem can be expressed as the probabilistic program in Fig. 1. The program has a loop that samples a die until it shows 6, and conditions on the number being even. In each iteration, the counter *Throws* is incremented.

1.1 Challenges

Bayesian inference. In Bayesian inference, Bayes' rule is used to update prior distributions $p(\theta)$ of model variables θ with observed data x to obtain posterior distributions: $p(\theta \mid x) = \frac{p(x|\theta)p(\theta)}{p(x)}$. In practice, such Bayesian statistical models are too complex for manual calculations and inferring their posterior distribution is a key challenge in Bayesian statistics. There are two approaches: exact and approximate inference. *Exact inference* aims to find an exact representation of the posterior distribution. Such methods impose heavy restrictions on the supported probabilistic programs and do not usually scale well. Practitioners therefore mostly use *approximate methods* that do not aim to compute this distribution exactly, but rather to produce unbiased or consistent samples from it. If the probabilistic program does not contain conditioning, samples can simply be obtained by running the program. But with observations, program runs that violate the observations must be rejected. Since the likelihood of the observations is typically low, simple rejection sampling is inefficient, and thus practical samplers use more sophisticated techniques, such as Markov chain Monte Carlo. While more scalable, these approaches typically do not provide strong guarantees on the approximation error after a finite amount of time [Gelman et al. 2013, Section 11.5].

```
$Throws := 0; Die := 0;$
while $Die \neq 6$ {
    $Die \sim$ Uniform$\{1, \dots, 6\}$;
    observe $Die \in \{2, 4, 6\}$;
    $Throws$ += 1}
```

Fig. 1. A probabilistic program with a loop and conditioning

Loops. Loops are essential to the expressiveness of programming languages but notoriously hard to analyze. This applies even more strongly to the probabilistic setting, where deciding properties like termination is harder than in the deterministic setting [Kaminski and Katoen 2015]. Even if a

[1]In a survey on Gil Kalai's blog, only 27% of participants chose the correct answer (https://gilkalai.wordpress.com/2017/09/07/tyi-30-expected-number-of-dice-throws/).

program does not use conditioning, loops can still make sampling difficult. For example, a program may terminate almost surely, but its expected running time may be infinite. This prevents sampling-based approaches since they need to run the program. Furthermore, many inference algorithms are not designed to handle unbounded loops and may return erroneous results for such programs [Beutner et al. 2022]. On the formal methods side, various approaches for probabilistic loop analysis have been proposed, employing techniques such as martingales, moments, and generating functions (see Section 7). If all program variables have finite support, the program can be translated to a probabilistic transition system and techniques from probabilistic model checking can be used.

None of these analysis techniques can be applied to Example 1.1 however: methods from cost analysis do not support conditioning and probabilistic model checking requires finite support (but *Throws* is supported on $\mathbb{N}$). The approach by Klinkenberg et al. [2024] via generating functions is theoretically applicable, but requires the user to provide a loop invariant template, i.e. a loop invariant where certain parameters may be missing. Unfortunately, such an invariant cannot always be specified in their language [Klinkenberg et al. 2024, Example 25]. Even in cases where this is possible, we argue that figuring out its shape is actually the hard part: it already requires a good understanding of the probabilistic program and its distribution, so it is not a satisfactory solution.

1.2 Guaranteed Bounds

To deal with the above challenges, we investigate ***guaranteed bounds*** on the program distribution. "Guaranteed" here refers to a method that computes deterministic (non-stochastic) results about the mathematical denotation of a program [Beutner et al. 2022]. Such bounds are applicable more often than exact inference, e.g. in the presence of loops/recursion, and provide more assurance than approximate methods, which have at best stochastic guarantees. Why are such bounds useful?

Partial correctness properties. In quantitative program analysis, one can verify safety properties by bounding the probability of reaching an unsafe state. Bounding reachability probabilities is also a common problem in probabilistic model checking and quantitative program verification, yet it has not seen much attention in the context of probabilistic programming with conditioning, aside from the work by Beutner et al. [2022] and Wang et al. [2024]. Neither of those can bound moments of infinite-support distributions, whereas our work finds tight bounds on the expected value of *Throws* in Fig. 1 (see Section 6.4).

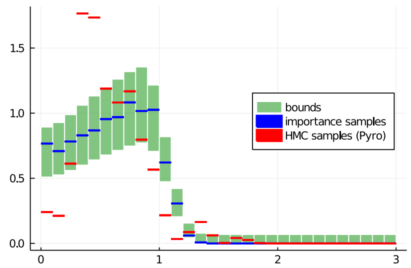


Fig. 2. Histogram of samples from two inference algorithms (importance sampling and Pyro's HMC), and the guaranteed bounds from Beutner et al. [2022]. The bounds show that Pyro's HMC produces wrong results. (Source: Beutner et al. [2022])

Checking approximate inference. In the context of Bayesian inference, the bounds can be useful to check and debug approximate inference algorithms and their implementations. If the approximate results clearly contradict the bounds, the inference algorithm is likely incorrect, or some of its assumptions are violated, or it has not converged. Beutner et al. [2022] provide an example of this: the inference tool Pyro yields wrong results for a probabilistic program with loops, but their bounds can detect this issue (Fig. 2).[2] Another problem with approximate inference is the tail behavior of the posterior

[2]The cause turned out to be an undocumented assumption in the inference algorithm. Pyro's implementation seems to assume that the dimension (number of samples in a program run) of the problem is constant, which is violated when sampling inside probabilistic loops.

Table 1. Comparison of our two approaches with the most relevant related work on probabilistic programs with loops. (Cond.: supports (Bayesian) conditioning; Inf.: allows branching on variables with infinite support; Cont.: allows continuous distributions; Auto.: fully automated; Prob.: computes/bounds probabilities; Mom.: computes/bounds moments; Tails: computes/bounds tail asymptotics of this shape; A.a. (always applicable): yields a result for any program expressible in the respective language. Partial support is denoted by "~".)

	Type	Cond.?	Inf.?	Cont.?	Auto.?	Prob.?	Mom.?	Tails?	A.a.?
Moosbrugger et al. [2022]	exact	✗	✗	✓	✓	~	✓	$O(n^{-k})$	✗
Beutner et al. [2022]	bounds	✓	✓	✓	✓	✓	✗	✗	✓
Wang et al. [2024]	bounds	✓	✓	✓	✓	✓	✗	✗	✗
Klinkenberg et al. [2024]	exact	✓	✓	✗	✗	✓	✓	✗	✗
Resid. mass sem. (Section 3)	bounds	✓	✓	✗	✓	✓	✗	✗	✓
Geom. bounds (Section 4)	bounds	✓	✓	✗	✓	✓	✓	$O(c^n)$	✗

distribution, which is often crucial for the quality of the approximation [Liang et al. 2023]. Previous work on guaranteed bounds [Beutner et al. 2022; Wang et al. 2024] does not address this aspect, but our work can bound the tail behavior as well.

Problem statement. Given a probabilistic program with posterior distribution μ on $\mathbb{N}$, our goal is to bound:

(1) *probability masses:* given $n \in \mathbb{N}$, find $l, u \in [0, 1]$ such that $l \leq \mathbb{P}_{X\sim\mu}[X = n] \leq u$;
(2) *moments:* given $k \in \mathbb{N}$, find $l, u \in \mathbb{R}_{\geq 0}$ such that $l \leq \mathbb{E}_{X\sim\mu}[X^k] \leq u$;
(3) *tail asymptotics:* find $c \in [0, 1)$ such that $\mathbb{P}_{X\sim\mu}[X = n] = O(c^n)$.

1.3 Contributions

In this paper, we develop two new methods to compute guaranteed bounds on the distribution of discrete probabilistic programs with loops and conditioning. Lower bounds can simply be found by unrolling each loop a finite number of times. The main challenge is upper bounds and we attack it in two ways: the first is simple, always applicable, and efficient, but coarse; the second is more sophisticated and expensive, but yields much more informative bounds if applicable. A summary of the most relevant related work is presented in Table 1 and a detailed account in Section 7.

The first semantics, called ***residual mass semantics*** (Section 3), is based on the simple idea of bounding the remaining probability mass after the loop unrollings, which has not previously been described, to our knowledge. We make the following contributions:

- We introduce the residual mass as a simple but effective idea to bound posterior probabilities.
- We prove soundness and convergence of the bounds to the true distribution (as loops are unrolled further and further).
- We implement the semantics in a tool called *Diabolo* and demonstrate empirically that the implementation is more efficient than previous systems (Section 6.3).

The second semantics, called ***geometric bound semantics*** (Section 4), is the main novelty of this paper. The idea is to bound the distribution of loops in a more fine-grained manner with geometric tails, rather than a flat bound as in the first semantics.

- We present the concept of a *contraction invariant* for a loop, which yields upper bounds on the distribution (Section 4.1).
- We introduce a family of distributions called *eventually geometric distributions (EGDs)* that are used as an abstract domain to overapproximate the distribution of a loop (Section 4.2).
- We present the *geometric bound semantics* (Section 4.3) which reduces the synthesis problem of such an invariant to solving a system of polynomial inequalities. If successful, it immediately

yields bounds on probability masses and, contrary to the first semantics, also on moments and tail probabilities of the program distribution.

- We prove soundness of the semantics and convergence of the bounds, as loops are unrolled further and further (Section 4.5).
- We identify necessary conditions and sufficient conditions for its applicability (Section 4.5).
- We fully automate it in our tool *Diabolo* (Section 5): contrary to previous work [Klinkenberg et al. 2024], it does not rely on the user to provide a loop invariant (template).
- We demonstrate its applicability on a large proportion of benchmarks from the literature and compare it with previous approaches and the residual mass semantics (Section 6).

Due to space constraints, proofs and some additional details had to be omitted. They can be found in the full version of this paper [Zaiser et al. 2024].

1.4 Limitations

Our work deals with *discrete* probabilistic programs with hard conditioning. This means that programs cannot sample or observe from continuous distributions. Variables in our programming language take values in $\mathbb{N}$; negative numbers are not supported (see Section 8.1 for possible extensions). While our language is Turing-complete, some arithmetic operations like multiplication as well as some common infinite-support distributions (e.g. Poisson) are not directly supported (see Section 2.2 for details on our language's expressivity). The initial values of the program variables are fixed: our methods cannot reason parametrically about these inputs.

The residual mass semantics can yield bounds on the distribution of any such probabilistic program, but convergence with increasing unrolling is only guaranteed if the program terminates almost surely. If the program distribution has infinite support, we cannot bound the moments or tails: the bound does not contain enough information for this.

The geometric bound semantics yields EGD bounds, which allow bounding moments and tails. On the other hand, such bounds do not exist for all programs. Our experiments show that this is not a big concern for many probabilistic programs with loops in practice: EGD bounds exist for a majority of examples we found in the literature. Another limitation of EGD bounds is that they cannot represent correlation of the tails of two variables, which may lead to imprecise tail bounds or failing to find bounds at all. Finally, solving the system of polynomial inequalities arising from the semantics, while decidable, can be hard in practice and does not scale to very large programs. It should be noted that scalability is a general issue in probabilistic program analysis owing to the hardness of the problem [Dagum and Luby 1993] and not specific to our work.

1.5 Notation and Conventions

We use the Iverson brackets $[\varphi]$ to mean 1 if φ is satisfied and 0 otherwise. We write variables representing vectors in bold ($\boldsymbol{\alpha}$), tensors (multidimensional arrays) in uppercase and bold ($\mathbf{T}$), and random or program variables in uppercase (X). We write $\mathbf{0}$ and $\mathbf{1}$ for the constant zero and one functions. We write $\mathbf{0}_n$ and $\mathbf{1}_n$ for the zero and ones vector in $\mathbb{R}^n$. To set the k-th component of $\boldsymbol{\alpha}$ to v, we write $\boldsymbol{\alpha}[k \mapsto v]$. Vectors $\boldsymbol{\alpha} \in \mathbb{R}^n$ are indexed as $\alpha_1, \dots, \alpha_n$. We abbreviate $[d] := \{0, \dots, d-1\}$. Tensors $\mathbf{T} \in \mathbb{R}^{[d_1]\times\cdots\times[d_n]}$ are indexed as $\mathbf{T}_{i_1,\dots,i_n}$ where i_k ranges from 0 to $d_k - 1$. We write $\mathbf{0}_{[d_1]\times\cdots\times[d_n]}$ or simply $\mathbf{0}$ for the zero tensor in $\mathbb{R}^{[d_1]\times\cdots\times[d_n]}$. We write $|\mathbf{T}| = (d_1 \dots, d_n)$ for the dimensions of $\mathbf{T} \in \mathbb{R}^{[d_1]\times\cdots\times[d_n]}$, and in particular $|\mathbf{T}|_i = d_i$. To index $\mathbf{T}$ along the k-th dimension, we write $\mathbf{T}_{k:j} \in \mathbb{R}^{[d_1]\times\cdots\times[d_{k-1}]\times[d_{k+1}]\times\cdots\times[d_n]}$, which is defined by $(\mathbf{T}_{k:j})_{i_1,\dots,i_{k-1},i_{k+1},\dots,i_n} = \mathbf{T}_{i_1,\dots,i_{k-1},j,i_{k+1},\dots,i_n}$. We often write tensor indices as $\boldsymbol{i} := (i_1, \dots, i_n)$ for brevity. We also abbreviate $\boldsymbol{\alpha}^{\boldsymbol{i}} := \prod_{k=1}^{n} \alpha_i^{i_k}$. Other binary operations (+, −, min, max, etc.) work elementwise on vectors and tensors, e.g. $(\boldsymbol{\alpha} + \boldsymbol{\beta})_j := \alpha_j + \beta_j$ and $\boldsymbol{\alpha} \le \boldsymbol{\beta}$ if and only if $\alpha_j \le \beta_j$ for all j.

2 Background

2.1 Probability and Measure Theory

All probability spaces M in this work are equipped with the discrete σ-algebra $\mathcal{P}(M)$. The set of measures on a space M is denoted by $\mathrm{Meas}(M)$. The ***restriction*** of a measure $\mu \in \mathrm{Meas}(M)$ to a set $M' \subseteq M$ is denoted by $\mu|_{M'}$. The ***mass function*** $m : M \to [0, 1]$ of a measure $\mu \in \mathrm{Meas}(M)$ is defined by $m(x) := \mu(\{x\})$. By abuse of notation, we will usually write μ for m as well. We will use the words ***distribution*** and ***measure*** interchangeably, but a ***probability distribution*** is a measure μ with ***total mass*** $\mu(M) = 1$. The k-th ***moment*** of a distribution $\mu \in \mathrm{Meas}(\mathbb{N})$ is defined as $\mathbb{E}_{X\sim\mu}[X^k] := \sum_{x\in\mathbb{N}} \mu(x) \cdot x^k$. The ***tail*** of a distribution refers to the region far away from the mean. A distribution μ on $\mathbb{N}$ has ***tail asymptotics*** $f(n)$ if $\mu(n) = \Theta(f(n))$. The $\mathsf{Dirac}(x)$ distribution has the mass function $\mathsf{Dirac}(x)(y) = [x = y]$; the $\mathsf{Bernoulli}(\rho)$ distribution for $\rho \in [0, 1]$ is given by $\mathsf{Bernoulli}(\rho)(0) = 1 - \rho$ and $\mathsf{Bernoulli}(\rho)(1) = \rho$; the $\mathsf{Uniform}\{a, \dots, b\}$ distribution for $a \le b \in \mathbb{N}$ by $\mathsf{Uniform}\{a, \dots, b\}(n) = \frac{[a\le n\le b]}{b-a+1}$; and the $\mathsf{Geometric}(\rho)$ distribution for $\rho \in (0, 1]$ by $\mathsf{Geometric}(\rho)(n) = (1 - \rho)^n \cdot \rho$.

2.2 Probabilistic Programming Language

Our programming language is a simple imperative language with a fixed number of variables $\boldsymbol{X} = (X_1, \dots, X_n)$. Each variable only takes values in $\mathbb{N}$. We consider the following minimal programming language where P denotes ***programs*** and E denotes ***events***.

$$P ::= \mathsf{skip} \mid P_1; P_2 \mid X_k := 0 \mid X_k \mathrel{+}= a \mid X_k \mathrel{\dot{-}}= 1 \mid \mathsf{if}\, E\, \{P_1\}\, \mathsf{else}\, \{P_2\} \mid \mathsf{while}\, E\, \{P_1\} \mid \mathsf{fail}$$
$$E ::= X_k = a \mid \mathsf{flip}(\rho) \mid \neg E_1 \mid E_1 \wedge E_2$$

where $a \in \mathbb{N}, \rho \in [0, 1]$ are literals. We explain the constructs briefly: skip does nothing; $P_1; P_2$ executes P_1 and then P_2; $X_k := 0$ sets X_k to 0; $X_k \mathrel{+}= a$ increments X_k by a; $X_k \mathrel{\dot{-}}= 1$ decrements X_k by 1 but clamped at 0 ($X_k := \max(X_k - 1, 0)$); $\mathsf{if}\, E\, \{P_1\}\, \mathsf{else}\, \{P_2\}$ executes P_1 if the event E occurs and P_2 otherwise; $\mathsf{while}\, E\, \{P_1\}$ repeats P_1 as long as E occurs; fail states a contradictory observation, i.e. $\mathsf{observe\ false}$. The $\mathsf{flip}(\rho)$ event occurs with probability ρ, like a coin flip with bias ρ coming up heads, or more formally, sampling 1 from an independent $\mathsf{Bernoulli}(\rho)$ distribution.[3] The logical operators $\neg, \wedge$ denote the complement and intersection of events. We usually assume that all variables are set to zero with probability 1 at the beginning.

Syntactic sugar. The language is kept minimal to simplify the presentation. The following syntactic sugar for events (on the left) and statements (on the right) will be convenient:

$$\begin{aligned} E_1 \vee E_2 &\rightsquigarrow \neg(\neg E_1 \wedge \neg E_2) \\ X_k \in \{a_1, \dots, a_m\} &\rightsquigarrow X_k = a_1 \vee \dots \vee X_k = a_m \\ X_k < a &\rightsquigarrow X_k \in \{0, \dots, a - 1\} \\ X_k \le a &\rightsquigarrow X_k < a + 1 \\ X_k \ge a &\rightsquigarrow \neg(X_k < a) \\ X_k > a &\rightsquigarrow \neg(X_k \le a) \end{aligned} \qquad \begin{aligned} X_k := c &\rightsquigarrow X_k := 0; X_k \mathrel{+}= c \\ \{P_1\}\ [\rho]\ \{P_2\} &\rightsquigarrow \mathsf{if\ flip}(\rho)\, \{P_1\}\, \mathsf{else}\, \{P_2\} \\ X_k \sim \mathsf{Bernoulli}(\rho) &\rightsquigarrow \{X_k := 1\}\ [\rho]\ \{X_k := 0\} \\ \mathsf{observe}\, E &\rightsquigarrow \mathsf{if}\, E\, \{\mathsf{skip}\}\, \mathsf{else}\, \{\mathsf{fail}\} \end{aligned}$$

Sampling. The above language does not include a sampling construct, but sampling from the following distributions can be encoded easily: all finite discrete distributions (e.g. Bernoulli, Binomial, Uniform, Categorical), the Geometric and Negative Binomial distributions, as well as shifted versions thereof. Finite discrete distributions can be expressed with branching and the $\mathsf{flip}(\rho)$ event, similarly

[3]This construct is not usually considered an event because it does not correspond to a subset of the state space. We preferred it to a separate sampling statement for convenience: it avoids auxiliary variables for conditionals and loops.

$$\begin{aligned}
\mu|_{X_k=a}(S) &:= \mu(\{\boldsymbol{x} \in S \mid x_k = a\}) \\
\mu|_{\mathsf{flip}(\rho)} &:= \rho \cdot \mu|_{\mathbb{N}^n} \\
\mu|_{\neg E_1} &:= \mu|_{\mathbb{N}^n} - \mu|_{E_1} \\
\mu|_{E_1 \wedge E_2} &:= (\mu|_{E_1})|_{E_2}
\end{aligned}$$

(a) Standard semantics of events

$$\begin{aligned}
[\![\mathsf{skip}]\!]_{\lightning}(\mu) &= \mu \\
[\![P_1; P_2]\!]_{\lightning}(\mu) &= [\![P_2]\!]_{\lightning}([\![P_1]\!]_{\lightning}(\mu)) \\
[\![X_k := 0]\!]_{\lightning}(\mu)(S) &= \mu|_{\lightning}(S) + \mu(\{\boldsymbol{x} \in \mathbb{N}^n \mid \boldsymbol{x}[k \mapsto 0] \in S\}) \\
[\![X_k \mathrel{+}= a]\!]_{\lightning}(\mu)(S) &= \mu|_{\lightning}(S) + \mu(\{\boldsymbol{x} \in \mathbb{N}^n \mid \boldsymbol{x}[k \mapsto x_k + a] \in S\}) \\
[\![X_k \mathrel{\dot{-}}= 1]\!]_{\lightning}(\mu)(S) &= \mu|_{\lightning}(S) + \mu(\{\boldsymbol{x} \in \mathbb{N}^n \mid \boldsymbol{x}[k \mapsto x_k \dot{-} 1] \in S\}) \\
[\![\mathsf{if}\, E\, \{P_1\}\, \mathsf{else}\, \{P_2\}]\!]_{\lightning}(\mu) &= \mu|_{\lightning} + [\![P_1]\!]_{\lightning}(\mu|_E) + [\![P_2]\!]_{\lightning}(\mu|_{\neg E}) \\
[\![\mathsf{fail}]\!]_{\lightning}(\mu) &= \mu(\mathbb{N}^n_{\lightning}) \cdot \mathsf{Dirac}(\lightning) \\
[\![\mathsf{while}\, E\, \{P_1\}]\!]_{\lightning} &= \mathsf{lfp}(\Phi_{E,P_1})
\end{aligned}$$

(b) Standard semantics of statements

Fig. 3. Standard semantics for probabilistic programs

to $\mathsf{Bernoulli}(\rho)$ shown above. The sampling construct $X_k \sim \mathsf{Geometric}(\rho)$ can be expressed as $X_k := 0; \mathsf{while}\ \neg\mathsf{flip}(\rho)\ \{X_k \mathrel{+}= 1\}$. A negative binomial distribution can be expressed similarly, as a sum of i.i.d. geometrically distributed variables. Sampling from other distributions, such as the Poisson distribution, can also be expressed in principle since our language is Turing-complete, but we are not aware of a simple encoding.

Expressivity. Our language is similar to the `cReDiP` language in Klinkenberg et al. [2024], but with more restricted sampling. Like `cReDiP`, our language does not support negative integers, continuous distributions/variables, more complex arithmetic like multiplication, or comparisons of two variables. Note that the non-probabilistic fragment of the language is already Turing-complete because it can simulate a three-counter machine. Thus these missing constructs could be encoded in principle, at the expense of program complexity.

2.3 Standard Semantics

The semantics of programs takes an input measure on the state space and yields an output measure. For probabilistic programs without conditioning, the state space of programs would be $\mathbb{N}^n$ [Klinkenberg et al. 2020; Zaiser et al. 2023]. But in the presence of conditioning, we want to track observations failures, so we add a ***failure state*** $\lightning$, which signifies a failed observation. If we just represented failures by setting the measure to 0 (as in Zaiser et al. [2023]), we would not be able to distinguish between observation failures and nontermination (see Klinkenberg et al. [2024] for a detailed discussion). So the semantics of programs transforms measures on the ***state space*** $\mathbb{N}^n_{\lightning} := \mathbb{N}^n \cup \{\lightning\}$. The intuitive idea is that when actually running a probabilistic program, fail moves to the failure state $\lightning$ and aborts the program.

Orders on measures and transformers. To define the semantics, we first need to define partial orders on measures and measure transformers, both of which are standard [Kozen 1981]. Given two measures μ, ν on M, we say that $\mu \le \nu$ if and only if $\mu(S) \le \nu(S)$ for all sets $S \subseteq M$. On measure transformers $\varphi, \psi : \mathrm{Meas}(M) \to \mathrm{Meas}(M)$, we define the pointwise lifting of this order: $\varphi \sqsubseteq \psi$ if and only if $\varphi(\mu) \le \psi(\mu)$ for all $\mu \in \mathrm{Meas}(M)$. Both orders are ω-complete partial orders.

Semantics. The ***standard semantics*** is a straightforward adaptation of Klinkenberg et al. [2024]; Kozen [1981]. For events E, it describes how a measure μ on program states is "restricted" to E (Fig. 3a), suggestively written $\mu|_E$ like the restriction of μ to a subset of $\mathbb{N}^n_{\lightning}$, even though some events (e.g. $\mathsf{flip}(\rho)$) do not correspond to such a subset. For programs P, the statement semantics $[\![P]\!]_{\lightning} : \mathrm{Meas}(\mathbb{N}^n_{\lightning}) \to \mathrm{Meas}(\mathbb{N}^n_{\lightning})$ describes how P transforms the measure on program states (Fig. 3b).

Regarding notation, we write $\mu|_{\lightning}$ for $\mu|_{\{\lightning\}}$, i.e. the restriction of μ to the failure state $\lightning$, and conversely $\mu|_{\mathbb{N}^n}$ for the restriction of μ to $\mathbb{N}^n$, excluding the failure state $\lightning$. In Section 4, we will largely ignore the failure state $\lightning$ and work with distributions on $\mathbb{N}^n$. For this purpose, we introduce the ***simplified standard semantics*** $[\![P]\!] : \mathrm{Meas}(\mathbb{N}^n) \to \mathrm{Meas}(\mathbb{N}^n)$ defined as $[\![P]\!](\mu) := [\![P]\!]_{\lightning}(\mu)|_{\mathbb{N}^n}$. An interesting case is the loop construct, whose semantics is $[\![\mathsf{while}\, E\, \{P_1\}]\!]_{\lightning} := \mathsf{lfp}(\Phi_{E,P_1})$ where

$$\Phi_{E,P_1} : (\mathrm{Meas}(\mathbb{N}^n_{\lightning}) \to \mathrm{Meas}(\mathbb{N}^n_{\lightning})) \to (\mathrm{Meas}(\mathbb{N}^n_{\lightning}) \to \mathrm{Meas}(\mathbb{N}^n_{\lightning}))$$
$$\Phi_{E,P_1}(\psi)(\mu) := \mu|_{\lightning} + \mu|_{\neg E} + \psi([\![P_1]\!]_{\lightning}(\mu|_E))$$

is the unrolling operator and $\mathsf{lfp}(\Phi_{E,P_1})$ denotes its least fixed point with respect to $\sqsubseteq$. Another way to look at it is that $W := [\![\mathsf{while}\, E\, \{P_1\}]\!]_{\lightning}$ is the least solution to the equation:

$$W(\mu) = \mu|_{\lightning} + \mu|_{\neg E} + W([\![P_1]\!]_{\lightning}(\mu|_E)) \quad \forall \mu \in \mathrm{Meas}(\mathbb{N}^n_{\lightning})$$

Conditioning. The fail construct moves all the mass to the failure state $\lightning$. Over the course of the program, some of the initial probability mass gets lost due to nontermination and some moves to the failure state due to failed observations. Ultimately, we are interested in the distribution *conditioned* on the observations. These conditional probabilities of $\boldsymbol{X} = \boldsymbol{x} \in \mathbb{N}^n$ are defined as: $\mathbb{P}[\boldsymbol{X} = \boldsymbol{x} \mid \boldsymbol{X} \neq \lightning] = \frac{\mathbb{P}[\boldsymbol{X}=\boldsymbol{x} \wedge \boldsymbol{X} \neq \lightning]}{\mathbb{P}[\boldsymbol{X} \neq \lightning]} = \frac{\mathbb{P}[\boldsymbol{X}=\boldsymbol{x}]}{1-\mathbb{P}[\boldsymbol{X}=\lightning]}$. Thus we have to remove the mass on $\lightning$ from the subprobability measure μ of the program and ***normalize*** it, which is achieved by the function normalize turning subprobability measures on $\mathbb{N}^n_{\lightning}$ into subprobability measures on $\mathbb{N}^n$: $\mathrm{normalize}(\mu) := \frac{\mu - \mu|_{\lightning}}{1-\mu(\lightning)}$. Thus the posterior probability distribution of a program P with initial distribution μ (usually $\mathsf{Dirac}(\mathbf{0}_n)$) is given by $\mathrm{normalize}([\![P]\!]_{\lightning}(\mu))$. Operationally, we can think of normalization as a rejection sampler of the posterior distribution: it runs the program repeatedly, rejects all runs that end in $\lightning$, and only yields the results of the remaining runs as samples.

Nontermination. Due to the possibility of nontermination, even the normalized measure of a program may not be a probability distribution, only a subprobability distribution (see the discussion in Klinkenberg et al. [2024, Section 3.4]). For example, the loop $\mathsf{while}\,\mathsf{flip}(1)\,\{\mathsf{skip}\}$ will never terminate, so both its unnormalized and normalized semantics are always the zero measure. This is not a problem in practice because nontermination is usually considered a bug for statistical models. Tracking observation failures is useful for defining termination, however.

Definition 2.1. A program P is ***almost surely terminating*** on a *finite* measure $\mu \in \mathrm{Meas}(\mathbb{N}^n_{\lightning})$ if $[\![P]\!]_{\lightning}(\mu)(\mathbb{N}^n_{\lightning}) = \mu(\mathbb{N}^n_{\lightning})$.

The semantics of a program $[\![P]\!]_{\lightning}$ satisfies the usual properties listed below. The proof is analogous to Klinkenberg et al. [2024].

Lemma 2.2. *For any program P, the transformation $[\![P]\!]_{\lightning}$ is linear and ω-continuous, so in particular monotonic. Also, the total measure does not increase: $[\![P]\!]_{\lightning}(\mu)(\mathbb{N}^n_{\lightning}) \leq \mu(\mathbb{N}^n_{\lightning})$ for all $\mu \in \mathrm{Meas}(\mathbb{N}^n_{\lightning})$.*

3 Residual Mass Semantics

In this section, we first present the *lower bound semantics* based on loop unrolling. Since even upper bounds on the normalized posterior require lower bounds on the normalizing constant, these lower bounds are also used as part of our methods for upper bounds. The *residual mass semantics* extends the lower bound semantics to obtain upper bounds on the unnormalized distribution as well and can thus derive both lower and upper bounds on the normalized posterior. This way, any exact inference method for the loop-free fragment can be transformed into a method to bound the distribution of programs with loops. We present both semantics formally and prove soundness and convergence. Full proofs are given in Zaiser et al. [2024].

3.1 Lower Bounds via Unrolling

Lower bounds can be computed by unrolling the loop a finite number of times and discarding the part of the distribution that has not exited the loop after the unrollings. Define the ***u*-fold unrolling** $P^{(u)}$ of a program P by unrolling each loop u times:

$$\begin{aligned}
(P_1; P_2)^{(u)} &:= P_1^{(u)}; P_2^{(u)} \\
(\mathsf{if}\ E\ \{P_1\}\ \mathsf{else}\ \{P_2\})^{(u)} &:= \mathsf{if}\ E\ \{P_1^{(u)}\}\ \mathsf{else}\ \{P_2^{(u)}\} \\
(\mathsf{while}\ E\ \{P\})^{(u)} &:= (\mathsf{while}\ E\ \{P\})_0^{(u)} \\
(\mathsf{while}\ E\ \{P\})_v^{(u)} &:= \mathsf{if}\ E\ \{P^{(u)}; (\mathsf{while}\ E\ \{P\})_{v+1}^{(u)}\}\ \mathsf{else}\ \{\mathsf{skip}\}; \quad \text{for } v < u \\
(\mathsf{while}\ E\ \{P\})_u^{(u)} &:= \mathsf{while}\ E\ \{P^{(u)}\} \\
P^{(u)} &:= P \quad \text{otherwise}
\end{aligned}$$

Lemma 3.1. *Unrolling does not change the semantics: $[\![P]\!]_{\lightning} = [\![P^{(u)}]\!]_{\lightning}$ for all programs P and $u \in \mathbb{N}$.*

The ***lower bound semantics*** $[\![P]\!]_{\mathrm{lo}}$ is then defined just like $[\![P]\!]_{\lightning}$ except that $[\![\mathsf{while}\ E\ \{P_1\}]\!]_{\mathrm{lo}}(\mu) := \mathbf{0}$. In other words, $[\![P]\!]_{\mathrm{lo}} = [\![P']\!]_{\lightning}$ where P' is the program P where all loops are replaced by infinite loops, effectively setting the measure to zero. Its correctness follows from the monotonicity of the standard semantics.

Theorem 3.2 (Soundness of lower bounds)**.** *For all measures $\mu \in \mathrm{Meas}(\mathbb{N}_{\lightning}^n)$ and programs P, we have $[\![P]\!]_{\mathrm{lo}}(\mu) \leq [\![P]\!]_{\lightning}(\mu)$.*

The lower bounds will get better as loops are unrolled further and further. In fact, they will converge to the true distribution.

Theorem 3.3 (Convergence of lower bounds)**.** *For all* finite *measures $\mu \in \mathrm{Meas}(\mathbb{N}_{\lightning}^n)$ and programs P, the lower bound $[\![P^{(u)}]\!]_{\mathrm{lo}}(\mu)$ converges (in total variation distance) monotonically to the true distribution $[\![P]\!]_{\lightning}(\mu)$ as $u \to \infty$.*

Note that the lower bound semantics involves only finite discrete distributions (assuming all variables are initially zero), which can be represented exactly as an array of the probability masses. In fact, any exact semantics for the loop-free fragment leads to lower bounds for programs with loops. A related approach by Jansen et al. [2016] combines unrolling with probabilistic model checking to obtain lower bounds on reachability probabilities and expectations.

3.2 Upper Bounds via Residual Mass

Lower bounds are easy because $\mathbf{0}$ is clearly a lower bound that can be used as a starting point for the fixed point iteration. This strategy does not work for upper bounds since there is no such obvious starting point.

Instead, a simple idea is to use the fact that the total mass of the state distribution can only decrease after program execution (Lemma 2.2). In other words, for a program P with initial distribution $\mu \in \mathrm{Meas}(\mathbb{N}_{\lightning}^n)$, the total mass of the distribution after running P is $[\![P]\!]_{\lightning}(\mu)(\mathbb{N}_{\lightning}^n) \leq \mu(\mathbb{N}_{\lightning}^n)$. So the part of the distribution that we miss in the lower bounds by cutting loops short has a probability mass bounded by the distance to the total mass $\mu(\mathbb{N}_{\lightning}^n)$ at the start. We call this gap the ***residual mass*** of the program P with initial measure μ and define it as $[\![P]\!]_{\mathrm{res}}(\mu) := \mu(\mathbb{N}_{\lightning}^n) - [\![P]\!]_{\mathrm{lo}}(\mu)(\mathbb{N}_{\lightning}^n) \in \mathbb{R}_{\geq 0}$.

The ***residual mass semantics*** uses the residual mass to bound probabilities $[\![P]\!]_{\lightning}(\mu)(S)$ of the program distribution for a set $S \subseteq \mathbb{N}_{\lightning}^n$ by analyzing the gap $[\![P]\!]_{\lightning}(\mu)(S) - [\![P]\!]_{\mathrm{lo}}(\mu)(S)$ as follows. The gap is maximized if $S = \mathbb{N}^n_{\lightning}$ and thus bounded by $[\![P]\!]_{\lightning}(\mu)(\mathbb{N}^n_{\lightning}) - [\![P]\!]_{\mathsf{lo}}(\mu)(\mathbb{N}^n_{\lightning})$. Since $[\![P]\!]_{\lightning}(\mu)(\mathbb{N}^n_{\lightning}) \leq \mu(\mathbb{N}^n_{\lightning})$ by Lemma 2.2, this gap is always bounded by the residual mass. The residual mass semantics can also bound the normalizing constant $1 - [\![P]\!]_{\lightning}(\mu)(\lightning)$ since $[\![P]\!]_{\mathsf{lo}}(\mu)(\lightning) \leq [\![P]\!]_{\lightning}(\mu)(\lightning) \leq [\![P]\!]_{\mathsf{lo}}(\mu)(\lightning) + [\![P]\!]_{\mathsf{res}}(\mu)$ where the lower bound follows from Theorem 3.2 and the upper bound from the previous argument. Thus it also yields bounds on the (normalized) posterior probabilities. The above soundness argument is made fully rigorous in the following theorem. We also have a convergence theorem for the residual mass semantics, similar to Theorem 3.3.

Theorem 3.4 (Soundness of the residual mass semantics)**.** *Let P be a program and $\mu \in \mathrm{Meas}(\mathbb{N}^n_{\lightning})$. Then for all $S \subseteq \mathbb{N}^n_{\lightning}$, we can bound the unnormalized probabilities:*

$$[\![P]\!]_{\mathsf{lo}}(\mu)(S) \leq [\![P]\!]_{\lightning}(\mu)(S) \leq [\![P]\!]_{\mathsf{lo}}(\mu)(S) + [\![P]\!]_{\mathsf{res}}(\mu)$$

If μ is a probability measure and $S \subseteq \mathbb{N}^n$, we can bound the posterior probabilities:

$$\frac{[\![P]\!]_{\mathsf{lo}}(\mu)(S)}{1 - [\![P]\!]_{\mathsf{lo}}(\mu)(\lightning)} \leq \mathrm{normalize}([\![P]\!]_{\lightning}(\mu))(S) \leq \frac{[\![P]\!]_{\mathsf{lo}}(\mu)(S) + [\![P]\!]_{\mathsf{res}}(\mu)}{1 - [\![P]\!]_{\mathsf{lo}}(\mu)(\lightning) - [\![P]\!]_{\mathsf{res}}(\mu)}$$

Theorem 3.5 (Convergence of the residual mass semantics)**.** *Let P be a program terminating almost surely on a* finite *measure $\mu \in \mathrm{Meas}(\mathbb{N}^n_{\lightning})$. Then the residual mass $[\![P^{(u)}]\!]_{\mathsf{res}}(\mu)$ converges monotonically to 0 as $u \to \infty$. In particular, the above bounds on* $\mathrm{normalize}([\![P^{(u)}]\!]_{\lightning}(\mu))(S)$ *converge to the true posterior probability as $u \to \infty$.*

Since the residual mass semantics only depends on the lower bounds, it can be implemented using the same representations and techniques as the lower bound semantics. In particular, it also involves only finite discrete distributions, representable as arrays of probability masses. We illustrate the residual mass semantics with an example.

Example 3.6 (Residual mass semantics)**.** Consider again the "die paradox" program (Example 1.1 and Fig. 1) where the variables are $\boldsymbol{X} := (\mathit{Throws}, \mathit{Die})$. After three unrollings, the lower bound distribution $\mu := [\![P^{(3)}]\!]_{\mathsf{lo}}(\mu_0)$ for the initial distribution $\mu_0 = \mathrm{Dirac}(\mathbf{0})$ is given by

$$\mu(\boldsymbol{x}) = \begin{cases} \frac{1}{2} + \frac{1}{3} \cdot \frac{1}{2} = \frac{2}{3} & \text{if } \boldsymbol{x} = \lightning \\ \frac{1}{6} & \text{if } \boldsymbol{x} = (1,6) \\ \frac{1}{3} \cdot \frac{1}{6} = \frac{1}{18} & \text{if } \boldsymbol{x} = (2,6) \\ 0 & \text{otherwise} \end{cases}$$

because the failure probability is $\frac{1}{2}$ (for $\mathit{Die} \in \{1,3,5\}$) and the second iteration is entered in a state other than $\lightning$ with probability $\frac{1}{3}$ (for $\mathit{Die} \in \{2,4\}$). The residual mass is given by $[\![P^{(3)}]\!]_{\mathsf{res}}(\mu_0) = \mu_0(\mathbb{N}^n_{\lightning}) - \mu(\mathbb{N}^n_{\lightning}) = 1 - (\frac{2}{3} + \frac{1}{6} + \frac{1}{18}) = \frac{1}{9}$. This yields the following bounds on $[\![P]\!]_{\lightning}(\mu_0)(s)$ for $s \in \mathbb{N}^n_{\lightning}$:

$$[\![P]\!]_{\lightning}(\mu_0)(\boldsymbol{x}) \in \begin{cases} [\frac{2}{3}, \frac{7}{9}] & \text{if } \boldsymbol{x} = \lightning \\ [\frac{1}{6}, \frac{5}{18}] & \text{if } \boldsymbol{x} = (1,6) \\ [\frac{1}{18}, \frac{1}{6}] & \text{if } \boldsymbol{x} = (2,6) \\ [0, \frac{1}{9}] & \text{otherwise} \end{cases}$$

Hence the normalizing constant $1 - [\![P]\!]_{\lightning}(\mu_0)$ is in $[\frac{2}{9}, \frac{1}{3}]$, which yields the following normalized bounds. They are not very tight but can be improved by increasing the unrolling depth.

$$\mathrm{normalize}([\![P]\!]_{\lightning}(\mu_0))(\boldsymbol{x}) \in \begin{cases} [\frac{1}{6} \cdot 3, \frac{5}{18} \cdot \frac{9}{2}] = [\frac{1}{2}, \frac{5}{4}] & \text{if } \boldsymbol{x} = (1,6) \\ [\frac{1}{18} \cdot 3, \frac{1}{6} \cdot \frac{9}{2}] = [\frac{1}{6}, \frac{3}{4}] & \text{if } \boldsymbol{x} = (2,6) \\ [0, \frac{1}{9} \cdot \frac{9}{2}] = [0, \frac{1}{2}] & \text{otherwise} \end{cases}$$

4 Geometric Bound Semantics

A big problem with the residual mass semantics is that we can bound the total measure and individual probabilities, but not moments and tail distributions of distributions with infinite support. This problem cannot be fully resolved in general because moments do not always exist.

Example 4.1. The moments of the distribution of the following program are infinite:

```
$X := 0; Y := 1;$
while flip$(1/2)$ {
    while $Y > 0$ {$X$ += 1; $Y \dot{-}= 1$};
    while $X > 0$ {$X \dot{-}= 1$; $Y$ += 2}; }
```

The loop body multiplies Y by 2 using the auxiliary variable X. This program is almost surely terminating and with result probabilities $\mathbb{P}[X = 0, Y = 2^m] = 2^{-m}$ for $m > 0$ and zero elsewhere. Hence the k-th moment of Y is $\mathbb{E}[Y^k] = \sum_{m=1}^{\infty} \mathbb{P}[X = 0, Y = 2^m] \cdot 2^{km} \geq \sum_{m=1}^{\infty} 1 = \infty$ for $k \geq 1$.

However, in many cases, more precise upper bounds can be found that also yield bounds on moments and tails. There are two key ideas that make this *geometric bound semantics* possible: contraction invariants and eventually geometric distributions (EGDs). Like the residual mass semantics, it builds on the lower bound semantics to obtain bounds on the normalized posterior.

4.1 Contraction Invariants

If a loop $L = \text{while}\, E\, \{B\}$ terminates almost surely on an initial distribution μ, all the probability mass that entered the loop must eventually exit the loop. Inspired by this observation, we consider the following assumption:

$$[\![B]\!]_{\lightning}(\mu|_E) \leq c \cdot \mu \text{ with } 0 \leq c < 1 \tag{1}$$

In words: the probability distribution at the start of the next loop iteration decreases uniformly by a factor of c. This assumption looks very strong and is often violated in practice, but if we assume for a minute that it holds, what can we derive? Recall the fixpoint equation of loops:

$$\begin{aligned}[\![\text{while}\, E\, \{B\}]\!]_{\lightning}(\mu) &= \mu|_{\lightning} + \mu|_{\neg E} + [\![\text{while}\, E\, \{B\}]\!]_{\lightning}([\![B]\!]_{\lightning}(\mu|_E)) \\ &\leq \mu|_{\lightning} + \mu|_{\neg E} + c \cdot [\![\text{while}\, E\, \{B\}]\!]_{\lightning}(\mu) \qquad \text{by Eq. (1) and linearity}\end{aligned}$$

By rearranging, we find the upper bound $[\![\text{while}\, E\, \{B\}]\!]_{\lightning}(\mu) \leq \frac{\mu|_{\lightning} + \mu|_{\neg E}}{1-c}$.

Unfortunately, the initial distribution μ will usually not satisfy Eq. (1) directly. But we may be able to increase μ to $\nu \geq \mu$ such that ν satisfies $[\![B]\!]_{\lightning}(\nu|_E) \leq c \cdot \nu$ with $0 \leq c < 1$. We then call ν a ***contraction invariant*** or c-***contraction invariant*** of μ.[4] Surprisingly, such a contraction invariant ν can often be found in practice (see Section 6.1), and we can then derive the following upper bound on the distribution after the loop: $[\![\text{while}\, E\, \{B\}]\!]_{\lightning}(\mu) \leq [\![\text{while}\, E\, \{B\}]\!]_{\lightning}(\nu) \leq \frac{\nu|_{\lightning} + \nu|_{\neg E}}{1-c}$.

4.2 Eventually Geometric Distributions (EGDs)

How can we find such a contraction invariant ν? We clearly have to restrict the candidate set for ν in some way. Specifically, we consider a class of distributions that generalize geometric distributions. Recall that a Geometric(β) distribution has probability masses $p(i) = \beta(1-\beta)^i \propto (1-\beta)^i$ with a decay rate of $1 - \beta$. For the multivariate setting, we consider products of independent geometric distributions, but with a twist: probabilities of small values are allowed to differ for additional flexibility. This section makes extensive use of vector and tensor notations as described in Section 1.5.

[4]The name is inspired by contraction mappings in mathematics where they describe functions $f : X \to Y$ of metric spaces with $d(f(x), f(y)) \leq c \cdot d(x, y)$ with $c < 1$.

Definition 4.2. Let $\mathbf{P} \in \mathbb{R}_{\geq 0}^{[d_1+1]\times\cdots\times[d_n+1]}$ be an n-dimensional tensor and $\boldsymbol{\alpha} \in [0,1)^n$. The n-dimensional ***eventually geometric distribution (EGD)*** with ***initial block*** P and ***decay rates*** $\boldsymbol{\alpha}$, written $\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha}) \in \mathrm{Meas}(\mathbb{N}^n)$ is defined by the following mass function (for $\boldsymbol{i} = (i_1, \dots, i_n) \in \mathbb{N}^n$):

$$\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha})(\boldsymbol{i}) = \mathbf{P}_{\min(\boldsymbol{i}, |\mathbf{P}|-\mathbf{1}_n)} \cdot \boldsymbol{\alpha}^{\max(\boldsymbol{i}-|\mathbf{P}|+\mathbf{1}_n, \mathbf{0}_n)} = \mathbf{P}_{\min(i_1,d_1),\dots,\min(i_n,d_n)} \alpha_1^{\max(i_1-d_1,0)} \cdots \alpha_n^{\max(i_n-d_n,0)}$$

Note that an EGD is not necessarily a probability measure. One can think of $\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha})$ as a scaled product of independent $\mathsf{Geometric}(1-\alpha_i)$ distributions, except it may differ in a finite "prefix" of size d_j in each dimension j. Outside this initial block, the probability masses are extended like in a geometric distribution with decay rate α_j in each dimension j. In other words: *eventually*, for values $i_j > d_j$, the j-th component of $\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha})$ behaves like a $\mathsf{Geometric}(1-\alpha_j)$ distribution; hence the name. EGDs were originally motivated by the shape of their generating function [Zaiser 2024b, Appendix C.1]. They can be seen as a subclass of multivariate discrete phase-type distributions [Campillo Navarro 2018]. (Discrete phase-type distributions [Neuts 1975] describe the absorption time in a Markov chain with one absorbing state [Bladt and Nielsen 2017, Section 1.2.6].)

Example 4.3. Consider the two-dimensional EGD given by $\mathsf{EGD}(\mathbf{P}, (\alpha_1, \alpha_2))$ with $\mathbf{P} \in \mathbb{R}_{\geq 0}^{[2]\times[2]}$, i.e. $\mathbf{P}$ is a matrix $\mathbf{P} = \begin{pmatrix} \mathbf{P}_{0,0} & \mathbf{P}_{0,1} \\ \mathbf{P}_{1,0} & \mathbf{P}_{1,1} \end{pmatrix}$. Its probability masses are given in the following table:

$\mathsf{EGD}(\mathbf{P}, (\alpha_1, \alpha_2))(x_1, x_2)$	$x_2 = 0$	$x_2 = 1$	$x_2 = 2$	$\dots$	$x_2 = j$
$x_1 = 0$	$\mathbf{P}_{0,0}$	$\mathbf{P}_{0,1}$	$\mathbf{P}_{0,1} \cdot \alpha_2$	$\dots$	$\mathbf{P}_{0,1} \cdot \alpha_2^{j-1}$
$x_1 = 1$	$\mathbf{P}_{1,0}$	$\mathbf{P}_{1,1}$	$\mathbf{P}_{1,1} \cdot \alpha_2$	$\dots$	$\mathbf{P}_{1,1} \cdot \alpha_2^{j-1}$
$x_1 = 2$	$\mathbf{P}_{1,0} \cdot \alpha_1$	$\mathbf{P}_{1,1} \cdot \alpha_1$	$\mathbf{P}_{1,1} \cdot \alpha_1 \alpha_2$	$\dots$	$\mathbf{P}_{1,1} \cdot \alpha_1 \cdot \alpha_2^{j-1}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$x_1 = i$	$\mathbf{P}_{1,0} \cdot \alpha_1^{i-1}$	$\mathbf{P}_{1,1} \cdot \alpha_1^{i-1}$	$\mathbf{P}_{1,1} \cdot \alpha_1^{i-1} \cdot \alpha_2$	$\dots$	$\mathbf{P}_{1,1} \cdot \alpha_1^{i-1} \cdot \alpha_2^{j-1}$

Here we can see that after the initial block of size 2×2, the probability masses are extended with decay rate α_1 in the first dimension and α_2 in the second dimension.

Why did we pick EGDs as the shape of our bounds? There are several reasons.

(1) *Interpretability:* EGDs can easily be understood as geometric distributions, where the probability masses for small values (up to some threshold) have been modified. The relationship with the geometric distribution also makes it easier to compute its moments and the tail asymptotics can be read directly off the parameter $\boldsymbol{\alpha}$ (see Theorem 4.5 below).
(2) *Expressiveness:* the reason we allow the "start" of the distribution to deviate from geometric distributions is necessary for flexibility. If only exact geometric distributions were allowed, typical program operations like increasing or decreasing a variable (i.e. shifting the geometric distribution) could not be represented precisely enough and the whole approach would fail.
(3) *Tractability:* a sufficient condition for the order $\leq$ is easy to check for EGDs, as we will see, because the probability masses follow a simple pattern. In fact, this condition is equivalent to the satisfiability of a system of polynomial inequalities. If we had based our semantics on an extension of, say, negative binomial distributions (of which geometric distributions are a special case), deciding the order would be harder because binomial coefficients start appearing in the probability masses.

As evidence for interpretability and tractability, we show how to marginalize EGDs and how to compute their moments. This is also needed to extract bounds on the moments from EGD bounds.

Lemma 4.4 (Marginalizing EGDs). *Marginalizing out the k-th dimension from $\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha})$ yields $\mathsf{EGD}(\mathbf{Q}, \boldsymbol{\beta})$ with $\boldsymbol{\beta} = (\alpha_1, \dots, \alpha_{k-1}, \alpha_{k+1}, \dots \alpha_n)$ and $\mathbf{Q} = \sum_{j=0}^{|\mathbf{P}|_k-2} \mathbf{P}_{k:j} + \frac{\mathbf{P}_{k:|\mathbf{P}|_k-1}}{1-\alpha_k}$.*

Theorem 4.5 (Moments and tails of EGDs). *Let* $\mathsf{EGD}(\mathbf{P}, \alpha)$ *be a one-dimensional EGD with* $\mathbf{P} \in \mathbb{R}_{\geq 0}^{[d+1]}$. *Then its tail asymptotics is* $\mathsf{EGD}(\mathbf{P}, \alpha)(n) = O(\alpha^n)$ *and its k-th moment is*

$$\mathbb{E}_{X \sim \mathsf{EGD}(\mathbf{P},\alpha)}[X^k] = \sum_{j=0}^{d-1} \mathbf{P}_j \cdot j^k + \frac{\mathbf{P}_d}{1-\alpha} \sum_{i=0}^{k} \binom{k}{i} d^{k-i} \mathbb{E}_{Y \sim \mathsf{Geometric}(1-\alpha)}[Y^i]$$

and can thus be computed from the k-th moment of a geometric distribution. In particular, the expected value is $\mathbb{E}_{X \sim \mathsf{EGD}(\mathbf{P},\alpha)}[X] = \sum_{j=0}^{d-1} \mathbf{P}_j \cdot j + \frac{\mathbf{P}_d}{1-\alpha}\left(d + \frac{\alpha}{1-\alpha}\right)$.

Expansion of EGDs. When performing binary operations (e.g. comparison or addition) on EGDs, it will often be convenient to assume that their initial blocks have the same size. In fact, one can always expand the size of the initial block of an EGD without changing the distribution. For example, the distribution from Example 4.3 can equivalently be represented as:

$$\mathsf{EGD}\left(\begin{pmatrix} \mathbf{P}_{0,0} & \mathbf{P}_{0,1} \\ \mathbf{P}_{1,0} & \mathbf{P}_{1,1} \end{pmatrix}, \boldsymbol{\alpha}\right) = \mathsf{EGD}\left(\begin{pmatrix} \mathbf{P}_{0,0} & \mathbf{P}_{0,1} \\ \mathbf{P}_{1,0} & \mathbf{P}_{1,1} \\ \mathbf{P}_{1,0}\alpha_1 & \mathbf{P}_{1,1}\alpha_1 \end{pmatrix}, \boldsymbol{\alpha}\right) = \mathsf{EGD}\left(\begin{pmatrix} \mathbf{P}_{0,0} & \mathbf{P}_{0,1} & \mathbf{P}_{0,1}\alpha_2 & \mathbf{P}_{0,1}\alpha_2^2 \\ \mathbf{P}_{1,0} & \mathbf{P}_{1,1} & \mathbf{P}_{1,1}\alpha_2 & \mathbf{P}_{1,1}\alpha_2^2 \end{pmatrix}, \boldsymbol{\alpha}\right)$$

More generally, given an $\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha})$, an ***expansion*** $\mathsf{EGD}(\mathbf{Q}, \boldsymbol{\alpha})$ with $|\mathbf{P}| \leq |\mathbf{Q}|$ (meaning $|\mathbf{P}|_i \leq |\mathbf{Q}|_i$ for all $i = 1, \dots, n$) is obtained by adding rows and columns with the appropriate factors of α_i, as follows:

$$\mathbf{Q}_{\boldsymbol{i}} = \mathbf{P}_{\min(\boldsymbol{i}, |\mathbf{P}|-\mathbf{1}_n)} \boldsymbol{\alpha}^{\max(\boldsymbol{i}-|\mathbf{P}|+\mathbf{1}_n, \mathbf{0}_n)} = \mathbf{P}_{\min(i_1, |\mathbf{P}|_1-1), \dots, \min(i_n, |\mathbf{P}|_n-1)} \alpha_1^{\max(i_1-|\mathbf{P}|_1+1, 0)} \cdots \alpha_n^{\max(i_n-|\mathbf{P}|_n+1, 0)}$$

Order of EGDs. To decide $\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha}) \preceq \mathsf{EGD}(\mathbf{Q}, \boldsymbol{\beta})$ for $\mathbf{P}$ and $\mathbf{Q}$ of the same size $|\mathbf{P}| = |\mathbf{Q}|$, one might hope that this would be equivalent to $\mathbf{P} \leq \mathbf{Q}$ and $\boldsymbol{\alpha} \leq \boldsymbol{\beta}$ (both elementwise), because such a simple conjunction of inequalities would be easy to check. To extend this idea to EGDs $\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha})$, $\mathsf{EGD}(\mathbf{Q}, \boldsymbol{\beta})$ of different sizes, we first have to expand them to the same size $\max(|\mathbf{P}|, |\mathbf{Q}|)$. Inlining the definition of expansion, we obtain the following order.

Definition 4.6 (Order of EGDs). The order $\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha}) \leq_{\mathsf{EGD}} \mathsf{EGD}(\mathbf{Q}, \boldsymbol{\beta})$ is defined to hold if and only if for all $\boldsymbol{i} < \max(|\mathbf{P}|, |\mathbf{Q}|)$ (using the notation explained in Section 1.5):

$$\boldsymbol{\alpha} \leq \boldsymbol{\beta} \ \wedge\ \mathbf{P}_{\min(\boldsymbol{i}, |\mathbf{P}|-\mathbf{1}_n)} \boldsymbol{\alpha}^{\max(\boldsymbol{i}-|\mathbf{P}|+\mathbf{1}_n, \mathbf{0}_n)} \leq \mathbf{Q}_{\min(\boldsymbol{i}, |\mathbf{Q}|-\mathbf{1}_n)} \boldsymbol{\beta}^{\max(\boldsymbol{i}-|\mathbf{Q}|+\mathbf{1}_n, \mathbf{0}_n)}$$

Example 4.7. Consider the $\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha})$ from Example 4.3 with $|\mathbf{P}| = (2, 2)$ and the two-dimensional $\mathsf{EGD}(\mathbf{Q}, \boldsymbol{\beta})$ with $|\mathbf{Q}| = (1, 4)$ and $\mathbf{Q} = \begin{pmatrix} \mathbf{Q}_{0,0} & \mathbf{Q}_{0,1} & \mathbf{Q}_{0,2} & \mathbf{Q}_{0,3} \end{pmatrix}$. Then $\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha}) \leq_{\mathsf{EGD}} \mathsf{EGD}(\mathbf{Q}, \boldsymbol{\beta})$ is equivalent to the following system of inequalities:

$$\begin{array}{lllll} \alpha_1 \leq \beta_1 & \mathbf{P}_{0,0} \leq \mathbf{Q}_{0,0} & \mathbf{P}_{0,1} \leq \mathbf{Q}_{0,1} & \mathbf{P}_{0,1} \cdot \alpha_2 \leq \mathbf{Q}_{0,2} & \mathbf{P}_{0,1} \cdot \alpha_2^2 \leq \mathbf{Q}_{0,3} \\ \alpha_2 \leq \beta_2 & \mathbf{P}_{1,0} \leq \mathbf{Q}_{0,0} \cdot \beta_1 & \mathbf{P}_{1,1} \leq \mathbf{Q}_{0,1} \cdot \beta_1 & \mathbf{P}_{1,1} \cdot \alpha_2 \leq \mathbf{Q}_{0,2} \cdot \beta_1 & \mathbf{P}_{1,1} \cdot \alpha_2^2 \leq \mathbf{Q}_{0,3} \cdot \beta_1 \end{array}$$

Unfortunately, this order $\leq_{\mathsf{EGD}}$ is not exactly the same as $\preceq$ because zero coefficients at the edge of $\mathbf{P}$ can make $\boldsymbol{\alpha}$ irrelevant, e.g. $\mathsf{EGD}(0, \alpha) = \mathbf{0}$ for any $\alpha \in [0, 1)$. Thus $\mathbf{0} = \mathsf{EGD}(0, \alpha) \preceq \mathsf{EGD}(1, 0) = \mathsf{Dirac}(0)$ even though $\mathsf{EGD}(0, \alpha) \not\leq_{\mathsf{EGD}} \mathsf{EGD}(1, 0)$ for $\alpha > 0$. However, such counterexamples are rare in practice, and $\leq_{\mathsf{EGD}}$ does imply $\preceq$, as the following lemma shows. As a consequence, working with the simpler order $\leq_{\mathsf{EGD}}$ instead of $\preceq$ is sufficient to establish bounds.

Lemma 4.8. *If* $\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha}) \leq_{\mathsf{EGD}} \mathsf{EGD}(\mathbf{Q}, \boldsymbol{\beta})$, *then* $\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha}) \preceq \mathsf{EGD}(\mathbf{Q}, \boldsymbol{\beta})$.

4.3 Semantics

Using the ideas from the previous sections, we define a compositional semantics $[\![P]\!]^{\text{geo}}$ for upper bounds on the unnormalized distribution of P. It is called ***geometric bound semantics*** because it operates on eventually geometric distribution bounds due to their desirable properties mentioned

Event E	Restriction $\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha})\|^{\text{geo}}_{E}$ of $\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha})$ to the event E
$X_k = a$	$\mathsf{EGD}(\mathbf{Q}, \boldsymbol{\alpha}[k \mapsto 0])$ where $\|\mathbf{Q}\| = \|\mathbf{P}\|[k \mapsto \max(\|\mathbf{P}\|_k, a + 2)]$ with $\mathbf{Q}_{k:j} = \mathbf{P}_{k:\min(j,\|\mathbf{P}\|_k-1)} \cdot \alpha_k^{\max(0,j-\|\mathbf{P}\|_k+1)} \cdot [j = a]$ for $j = 0, \dots, \|\mathbf{Q}\|_k - 1$
$\mathsf{flip}(\rho)$	$\mathsf{EGD}(\rho \cdot \mathbf{P}, \boldsymbol{\alpha})$
$\neg E_1$	$\mathsf{EGD}(\mathbf{Q}, \boldsymbol{\alpha})$ where $\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha})\|^{\text{geo}}_{E_1} = \mathsf{EGD}(\mathbf{R}, \boldsymbol{\gamma})$ $\|\mathbf{Q}\| = \|\mathbf{R}\|$ and $\mathbf{Q}_{\boldsymbol{i}} = \mathbf{P}_{\min(\boldsymbol{i},\|\mathbf{P}\|-\mathbf{1}_n)} \cdot \boldsymbol{\alpha}^{\max(\boldsymbol{i}-\|\mathbf{P}\|+\mathbf{1}_n,\mathbf{0}_n)} - \mathbf{R}_{\boldsymbol{i}} \quad \forall \boldsymbol{i} \le \|\mathbf{Q}\|$
$E_1 \wedge E_2$	$\left(\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha})\|^{\text{geo}}_{E_1}\right)\Big\|^{\text{geo}}_{E_2}$

Fig. 4. Geometric bound semantics for events

in Section 4.2. It also turns out that EGDs are closed under many operations we require: restricting to events, marginalizing, and adding or subtracting constants from variables.

The semantics $[\![P]\!]^{\text{geo}}$ operates on distributions on $\mathbb{N}^n$, not $\mathbb{N}^n_{\lightning}$. We do not track the failure state $\lightning$ in this semantics, because the geometric bound semantics is only applicable to almost surely terminating programs (see Theorem 4.16), so there is no need to distinguish between observation failure and nontermination. Like in the residual mass semantics, we can bound the normalizing constant with the help of the lower bound semantics (see Theorem 4.13).

Relational semantics. Since there may be more than one upper bound, $[\![P]\!]^{\text{geo}}$ is not a function, but a binary *relation* on EGDs. The idea is that $(\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha}), \mathsf{EGD}(\mathbf{Q}, \boldsymbol{\beta})) \in [\![P]\!]^{\text{geo}}$, also written $\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha})\ [\![P]\!]^{\text{geo}}\ \mathsf{EGD}(\mathbf{Q}, \boldsymbol{\beta})$, ensures that $[\![P]\!](\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha})) \preceq \mathsf{EGD}(\mathbf{Q}, \boldsymbol{\beta})$. If this $\mathsf{EGD}(\mathbf{Q}, \boldsymbol{\beta})$ is unique for a given $\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha})$, we will use function notation: $[\![P]\!]^{\text{geo}}(\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha})) = \mathsf{EGD}(\mathbf{Q}, \boldsymbol{\beta})$. Like $\preceq_{\text{EGD}}$, the relation $[\![P]\!]^{\text{geo}}$ depends on the *representations* $(\mathbf{P}, \boldsymbol{\alpha}), (\mathbf{Q}, \boldsymbol{\beta})$ of the involved EGDs $\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha})$ and $\mathsf{EGD}(\mathbf{Q}, \boldsymbol{\beta})$, not just their measures.

Event semantics. The event semantics $\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha})|^{\text{geo}}_{E}$ computes an EGD representation of the standard event semantics $\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha})|_E$ (Fig. 4). The restriction $\mathsf{EGD}(\mathbf{Q}, \boldsymbol{\beta}) := \mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha})|^{\text{geo}}_{E}$ is generally computed by expanding the initial block $\mathbf{P}$ (if necessary) and then zeroing its entries not corresponding to the event E. An important property is that $\mathsf{EGD}(\mathbf{Q}, \boldsymbol{\beta}) = \mathsf{EGD}(\mathbf{Q}, \boldsymbol{\alpha})$.

For the event $X_k = a$, we have to set the coefficients of $\mathbf{P}_{k:j}$ to zero for $j \neq a$. For this, we first have to expand $\mathbf{P}$ to $\mathbf{Q}$ with $|\mathbf{Q}|_k = a + 2$ if necessary. Then we set $\mathbf{Q}_{k:j} = \mathbf{0}$ for $j \neq a$ to ensure that all coefficients different from a are zero, and also set $\alpha_k = 0$. For instance, the EGD from Example 4.3 restricted to the event $X_2 = 2$ is

$$\mathsf{EGD}\left(\begin{pmatrix} \mathbf{P}_{0,0} & \mathbf{P}_{0,1} \\ \mathbf{P}_{1,0} & \mathbf{P}_{1,1} \end{pmatrix}, \boldsymbol{\alpha}\right)\Bigg|^{\text{geo}}_{X_2=2} = \mathsf{EGD}\left(\begin{pmatrix} 0 & 0 & \mathbf{P}_{0,1} \cdot \alpha_2 & 0 \\ 0 & 0 & \mathbf{P}_{1,1} \cdot \alpha_2 & 0 \end{pmatrix}, (\alpha_1, 0)\right)$$

Next, the event $\mathsf{flip}(\rho)$ is independent of the current distribution, so we can simply multiply the initial block $\mathbf{P}$ by the probability ρ. For the complement $\neg E_1$, we first compute the restriction $\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha})|^{\text{geo}}_{E_1} = \mathsf{EGD}(\mathbf{R}, \boldsymbol{\gamma})$. Since $\mathsf{EGD}(\mathbf{R}, \boldsymbol{\gamma}) = \mathsf{EGD}(\mathbf{R}, \boldsymbol{\alpha})$ by the property mentioned above, we can compute its complement $\mathsf{EGD}(\mathbf{Q}, \boldsymbol{\alpha}) := \mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha}) - \mathsf{EGD}(\mathbf{R}, \boldsymbol{\alpha})$ by first expanding $\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha})$ to $\mathsf{EGD}(\mathbf{P}', \boldsymbol{\alpha})$ with $\mathbf{P}'$ of the same size as $\mathbf{R}$, and then subtracting $\mathbf{Q} := \mathbf{P}' - \mathbf{R}$. So the result is $\mathsf{EGD}(\mathbf{Q}, \boldsymbol{\alpha})$. Continuing the above example with the event $\neg(X_2 = 2)$, we get:

$$\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha})|^{\text{geo}}_{\neg(X_2=2)} = \mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha}) - \mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha})|^{\text{geo}}_{X_2=2} = \mathsf{EGD}\left(\begin{pmatrix} \mathbf{P}_{0,0} & \mathbf{P}_{0,1} & 0 & \mathbf{P}_{0,1} \cdot \alpha_2^2 \\ \mathbf{P}_{1,0} & \mathbf{P}_{1,1} & 0 & \mathbf{P}_{1,1} \cdot \alpha_2^2 \end{pmatrix}, \boldsymbol{\alpha}\right)$$

Finally, the restriction to an intersection $E_1 \wedge E_2$ is computed by restricting to E_1 and then to E_2.

Statement P	Constraints for $\mathsf{EGD}(\mathbf{P},\boldsymbol{\alpha})\ \llbracket P\rrbracket^{\mathrm{geo}}\ \mathsf{EGD}(\mathbf{Q},\boldsymbol{\beta})$
skip	$\boldsymbol{\beta}=\boldsymbol{\alpha}\ \land\ \mathbf{Q}=\mathbf{P}$
$P_1; P_2$	$\exists\mathbf{R},\boldsymbol{\gamma}.\ \mathsf{EGD}(\mathbf{P},\boldsymbol{\alpha})\ \llbracket P_1\rrbracket^{\mathrm{geo}}\ \mathsf{EGD}(\mathbf{R},\boldsymbol{\gamma})\ \land\ \mathsf{EGD}(\mathbf{R},\boldsymbol{\gamma})\ \llbracket P_2\rrbracket^{\mathrm{geo}}\ \mathsf{EGD}(\mathbf{Q},\boldsymbol{\beta})$
$X_k := 0$	$\boldsymbol{\beta}=\boldsymbol{\alpha}[k\mapsto 0]\ \land\ \mathbf{Q}_{k:0}=\frac{\mathbf{P}_{k:\lvert\mathbf{P}\rvert_k-1}}{1-\alpha_k}+\sum_{j=0}^{\lvert\mathbf{P}\rvert_k-2}\mathbf{P}_{k:j}\ \land\ \mathbf{Q}_{k:1}=\mathbf{0}$ where $\lvert\mathbf{Q}\rvert=\lvert\mathbf{P}\rvert[k\mapsto 2]$
$X_k \mathrel{+}= a$	$\boldsymbol{\beta}=\boldsymbol{\alpha}\ \land\ \bigwedge_{j=0}^{\lvert\mathbf{Q}\rvert_k-1}\mathbf{Q}_{k:j}=\begin{cases}0 & \text{if } j<a\\ \mathbf{P}_{k:j-a} & \text{else}\end{cases}$ where $\lvert\mathbf{Q}\rvert=\lvert\mathbf{P}\rvert[k\mapsto\lvert\mathbf{P}\rvert_k+a]$
$X_k \mathbin{\dot{-}}= 1$	$\boldsymbol{\beta}=\boldsymbol{\alpha}\ \land\ \bigwedge_{j=0}^{\lvert\mathbf{Q}\rvert_k-1}\mathbf{Q}_{k:j}=\begin{cases}\mathbf{P}_{k:0}+\mathbf{P}_{k:\min(\lvert\mathbf{P}\rvert_k-1,1)}\cdot\alpha^{\max(2-\lvert\mathbf{P}\rvert_k,0)} & \text{if } j=0\\ \mathbf{P}_{k:\min(\lvert\mathbf{P}\rvert_k-1,j+1)}\cdot\alpha^{\max(j+2-\lvert\mathbf{P}\rvert_k,0)} & \text{else}\end{cases}$ where $\lvert\mathbf{Q}\rvert=\lvert\mathbf{P}\rvert[k\mapsto\max(\lvert\mathbf{P}\rvert_k-1,2)]$
if E $\{P_1\}$ else $\{P_2\}$	$\exists\mathbf{R},\mathbf{S},\boldsymbol{\gamma},\boldsymbol{\delta}.\ \mathsf{EGD}(\mathbf{P},\boldsymbol{\alpha})\rvert_E^{\mathrm{geo}}\ \llbracket P_1\rrbracket^{\mathrm{geo}}\ \mathsf{EGD}(\mathbf{R},\boldsymbol{\gamma})\ \land\ \mathsf{EGD}(\mathbf{P},\boldsymbol{\alpha})\rvert_{\neg E}^{\mathrm{geo}}\ \llbracket P_2\rrbracket^{\mathrm{geo}}\ \mathsf{EGD}(\mathbf{S},\boldsymbol{\delta})$ $\land\ (\mathsf{EGD}(\mathbf{R},\boldsymbol{\gamma}),\mathsf{EGD}(\mathbf{S},\boldsymbol{\delta}),\mathsf{EGD}(\mathbf{Q},\boldsymbol{\beta}))\in$ Join (see Definition 4.9)
while E $\{P\}$	$\exists c,\mathbf{R},\mathbf{S},\boldsymbol{\gamma},\boldsymbol{\delta}.\ 0\le c<1\ \land\ \mathsf{EGD}(\mathbf{P},\boldsymbol{\alpha})\preceq_{\mathsf{EGD}}\mathsf{EGD}(\mathbf{R},\boldsymbol{\gamma})$ $\land\ \mathsf{EGD}(\mathbf{R},\boldsymbol{\gamma})\rvert_E^{\mathrm{geo}}\ \llbracket P\rrbracket^{\mathrm{geo}}\ \mathsf{EGD}(\mathbf{S},\boldsymbol{\delta})\ \land\ \mathsf{EGD}(\mathbf{S},\boldsymbol{\delta})\preceq_{\mathsf{EGD}}\mathsf{EGD}(c\cdot\mathbf{R},\boldsymbol{\gamma})$ $\land\ \mathsf{EGD}\left(\frac{\mathbf{R}}{1-c},\boldsymbol{\gamma}\right)\Big\rvert_{\neg E}^{\mathrm{geo}}=\mathsf{EGD}(\mathbf{Q},\boldsymbol{\beta})$
fail	$\boldsymbol{\beta}=\mathbf{0}_n\ \land\ \mathbf{Q}=\mathbf{0}_{[1]\times\cdots\times[1]}$

Fig. 5. Geometric bound semantics for statements

Statement semantics. The statement semantics is a binary relation $\llbracket P\rrbracket^{\mathrm{geo}}$ on EGDs. It defines $\mathsf{EGD}(\mathbf{P},\boldsymbol{\alpha})\ \llbracket P\rrbracket^{\mathrm{geo}}\ \mathsf{EGD}(\mathbf{Q},\boldsymbol{\beta})$ by induction on the structure of P via constraints on $\mathbf{P},\boldsymbol{\alpha},\mathbf{Q},\boldsymbol{\beta}$ (Fig. 5) and ensures $\llbracket P\rrbracket(\mathsf{EGD}(\mathbf{P},\boldsymbol{\alpha}))\preceq\mathsf{EGD}(\mathbf{Q},\boldsymbol{\beta})$. The semantics of skip is the identity relation and $P_1; P_2$ is relational composition. Since fail corresponds to observing a zero-probability event, its right-hand side is a zero distribution. For $X_k := 0$, we essentially have to marginalize out the k-th dimension (see Lemma 4.4) and then put all the probability mass on $X_k = 0$. For instance, applying $\llbracket X_2 := 0\rrbracket^{\mathrm{geo}}$ to the $\mathsf{EGD}(\mathbf{P},\boldsymbol{\alpha})$ from Example 4.3, yields the unique right-hand side

$$\mathsf{EGD}(\mathbf{Q},\boldsymbol{\beta})=\mathsf{EGD}\left(\begin{pmatrix}\mathbf{P}_{0,0}+\frac{\mathbf{P}_{0,1}}{1-\alpha_2} & 0\\ \mathbf{P}_{1,0}+\frac{\mathbf{P}_{1,1}}{1-\alpha_2} & 0\end{pmatrix},(\alpha_1,0)\right)$$

For $X_k \mathrel{+}= c$, we shift the coefficients to the right by c in the k-th dimension, and fill up with zeros. For instance, applying $\llbracket X_2 \mathrel{+}= 2\rrbracket^{\mathrm{geo}}$ to the above EGD yields the unique right-hand side

$$\mathsf{EGD}(\mathbf{Q},\boldsymbol{\beta})=\mathsf{EGD}\left(\begin{pmatrix}0 & 0 & \mathbf{P}_{0,0} & \mathbf{P}_{0,1}\\ 0 & 0 & \mathbf{P}_{1,0} & \mathbf{P}_{1,1}\end{pmatrix},\boldsymbol{\alpha}\right)$$

For $X_k \mathbin{\dot{-}}= 1$, we shift the coefficients to the left by 1 in the k-th dimension, except for $\mathbf{P}_{k:0}$, which stays at index 0, so we get the sum of $\mathbf{P}_{k:0}$ and $\mathbf{P}_{k:1}$ at index 0. This special case requires $\lvert\mathbf{Q}\rvert_k\ge 2$, so we may first have to expand $\mathbf{P}$ to ensure $\lvert\mathbf{P}\rvert_k\ge 3$, which is done implicitly in Fig. 5. As an example, applying $\llbracket X_2 \mathbin{\dot{-}}= 1\rrbracket^{\mathrm{geo}}$ to the above EGD yields the unique right-hand side

$$\mathsf{EGD}(\mathbf{Q},\boldsymbol{\beta})=\mathsf{EGD}\left(\begin{pmatrix}\mathbf{P}_{0,0}+\mathbf{P}_{0,1} & \mathbf{P}_{0,1}\cdot\alpha_2\\ \mathbf{P}_{1,0}+\mathbf{P}_{1,1} & \mathbf{P}_{1,1}\cdot\alpha_2\end{pmatrix},\boldsymbol{\alpha}\right)$$

Conditionals. The semantics of if E $\{P_1\}$ else $\{P_2\}$ is more complex. We first require the existence of bounds on both branches ($\mathsf{EGD}(\mathbf{R},\boldsymbol{\gamma})$ and $\mathsf{EGD}(\mathbf{S},\boldsymbol{\delta})$), whose constraints are given by: $\mathsf{EGD}(\mathbf{P},\boldsymbol{\alpha})\rvert_E^{\mathrm{geo}}\ \llbracket P_1\rrbracket^{\mathrm{geo}}\ \mathsf{EGD}(\mathbf{R},\boldsymbol{\gamma})$ and $\mathsf{EGD}(\mathbf{P},\boldsymbol{\alpha})\rvert_{\neg E}^{\mathrm{geo}}\ \llbracket P_2\rrbracket^{\mathrm{geo}}\ \mathsf{EGD}(\mathbf{S},\boldsymbol{\delta})$. Finally, we would like to sum the bounds on the branches to obtain a bound on the whole conditional. However the sum of two EGDs may not be an EGD. Instead, we define a *join* $\mathsf{EGD}(\mathbf{Q}, \boldsymbol{\beta})$ as an upper bound on the sum $\mathsf{EGD}(\mathbf{R}, \boldsymbol{\gamma}) + \mathsf{EGD}(\mathbf{S}, \boldsymbol{\delta})$ of a certain shape.

Definition 4.9 (Join relation)**.** We say $\mathsf{EGD}(\mathbf{Q}, \boldsymbol{\beta})$ is a ***join*** of $\mathsf{EGD}(\mathbf{R}, \boldsymbol{\gamma})$ and $\mathsf{EGD}(\mathbf{S}, \boldsymbol{\delta})$, and write $(\mathsf{EGD}(\mathbf{R}, \boldsymbol{\gamma}), \mathsf{EGD}(\mathbf{S}, \boldsymbol{\delta}), \mathsf{EGD}(\mathbf{Q}, \boldsymbol{\beta})) \in \mathsf{Join}$, if $\boldsymbol{\gamma} \le \boldsymbol{\beta}$, $\boldsymbol{\delta} \le \boldsymbol{\beta}$ and there are expansions $\mathsf{EGD}(\mathbf{R}', \boldsymbol{\gamma})$ of $\mathsf{EGD}(\mathbf{R}, \boldsymbol{\gamma})$ and $\mathsf{EGD}(\mathbf{S}', \boldsymbol{\delta})$ of $\mathsf{EGD}(\mathbf{S}, \boldsymbol{\delta})$, of the same size, such that $\mathbf{Q} = \mathbf{R}' + \mathbf{S}'$. We also define the ***strict join*** relation $\mathsf{Join}^* \subset \mathsf{Join}$ that strengthens the condition $\boldsymbol{\beta} \ge \max(\boldsymbol{\gamma}, \boldsymbol{\delta})$ to $\boldsymbol{\beta} = \max(\boldsymbol{\gamma}, \boldsymbol{\delta})$.

As an example, the minimal-size join of $\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha})$ and $\mathsf{EGD}(\mathbf{Q}, \boldsymbol{\beta})$ from Example 4.7 is given by

$$\mathsf{EGD}\left(\begin{pmatrix} \mathbf{P}_{0,0} + \mathbf{Q}_{0,0} & \mathbf{P}_{0,1} + \mathbf{Q}_{0,1} & \mathbf{P}_{0,1} \cdot \alpha_2 + \mathbf{Q}_{0,2} & \mathbf{P}_{0,1} \cdot \alpha_2^2 + \mathbf{Q}_{0,3} \\ \mathbf{P}_{1,0} + \mathbf{Q}_{0,0} \cdot \beta_1 & \mathbf{P}_{1,1} + \mathbf{Q}_{0,1} \cdot \beta_1 & \mathbf{P}_{1,1} \cdot \alpha_2 + \mathbf{Q}_{0,2} \cdot \beta_1 & \mathbf{P}_{1,1} \cdot \alpha_2^2 + \mathbf{Q}_{0,3} \cdot \beta_1 \end{pmatrix}, (\gamma_1, \gamma_2)\right)$$

with $\alpha_1, \beta_1 \le \gamma_1 < 1$ and $\alpha_2, \beta_2 \le \gamma_2 < 1$. It is a strict join if $\gamma_1 = \max(\alpha_1, \beta_1)$ and $\gamma_2 = \max(\alpha_2, \beta_2)$. We use normal joins in the semantics because strict joins would introduce maxima in the constraints, making them harder to solve. Strict joins are useful for theoretical analysis, however.

Loops. Bounding a loop $\mathsf{while}\, E\, \{B\}$ requires the existence of a contraction invariant $\mathsf{EGD}(\mathbf{R}, \boldsymbol{\gamma})$ and a contraction factor $c \in [0, 1)$ (see Section 4.1), satisfying the following conditions. First, the initial distribution has to be bounded by the invariant: $\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha}) \preceq_{\mathsf{EGD}} \mathsf{EGD}(\mathbf{R}, \boldsymbol{\gamma})$. Second, the invariant has to decrease by a factor of c in each loop iteration, which is encoded as: there exists an $\mathsf{EGD}(\mathbf{S}, \boldsymbol{\delta})$ such that $\mathsf{EGD}(\mathbf{R}, \boldsymbol{\gamma})|_E^{\mathsf{geo}}\ [\![B]\!]^{\mathsf{geo}}\ \mathsf{EGD}(\mathbf{S}, \boldsymbol{\delta})$ and $\mathsf{EGD}(\mathbf{S}, \boldsymbol{\delta}) \preceq_{\mathsf{EGD}} \mathsf{EGD}(c \cdot \mathbf{R}, \boldsymbol{\gamma})$. Then $\mathsf{EGD}(\mathbf{Q}, \boldsymbol{\beta}) := \mathsf{EGD}\left(\frac{\mathbf{R}}{1-c}, \boldsymbol{\gamma}\right)\Big|_{\neg E}^{\mathsf{geo}}$ is an upper bound on $[\![\mathsf{while}\, E\, \{B\}]\!](\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha}))$, as discussed in Section 4.1.

Nondeterminism. Note that there are two places in the semantics where choices have to be made: conditionals and loops. This nondeterminism is the reason why the semantics $[\![-]\!]^{\mathsf{geo}}$ is a relation instead of a function. For $\mathsf{if}\, E\, \{P_1\}\, \mathsf{else}\, \{P_2\}$, the choice is in the join operation: concretely, the size of the expansion of the two distributions. For $\mathsf{while}\, E\, \{P\}$, the choice is in the contraction invariant $\mathsf{EGD}(\mathbf{R}, \boldsymbol{\gamma})$ and the factor c. First of all, the dimensions $|\mathbf{R}|$ have to be chosen. How we do this in practice is discussed in Section 5. Once the size of $\mathbf{R}$ is chosen, the conditions on $\mathbf{R}$ reduce to polynomial inequality constraints (a decidable problem). Of course, we typically want to find a "good" solution to these constraints by optimizing some objective (see Section 5.2).

4.4 Examples

Example 4.10 (Simple counter)**.** Consider a simple program representing a geometric distribution:

$$\mathsf{while}\, \mathsf{flip}(1/2)\, \{X_1 \mathrel{+}= 1\}$$

The starting state of the program is described by $\mathsf{EGD}(1, 0)$. Assume a c-contraction invariant $\mathsf{EGD}(p, \alpha)$ exists with $p \in \mathbb{R}_{\ge 0} \cong \mathbb{R}_{\ge 0}^{[1]}$ and $\alpha \in [0, 1)$. Then the constraint $\mathsf{EGD}(1, 0) \preceq_{\mathsf{EGD}} \mathsf{EGD}(p, \alpha)$ yields the inequalities $p \ge 1, \alpha \ge \mathbf{0}$.

Let the loop body be $B := (X_1 \mathrel{+}= 1)$. We find $\mathsf{EGD}(p, \alpha)|_{\mathsf{flip}(1/2)}^{\mathsf{geo}}\ [\![B]\!]^{\mathsf{geo}}\ \mathsf{EGD}\left(\begin{pmatrix} 0 & \frac{p}{2} \end{pmatrix}, \alpha\right)$. The constraint for the contraction invariant is $\mathsf{EGD}\left(\begin{pmatrix} 0 & \frac{p}{2} \end{pmatrix}, \alpha\right) \preceq_{\mathsf{EGD}} \mathsf{EGD}(c \cdot p, \alpha) = \mathsf{EGD}\left(\begin{pmatrix} cp & cp\alpha \end{pmatrix}, \alpha\right)$ and amounts to $p \ge 1$ (from above) and $\frac{1}{2} \le c \cdot \alpha$ with $c < 1$. The bound on the whole loop is then given by $\frac{1}{1-c}\ \mathsf{EGD}(p, \alpha)|_{\neg\mathsf{flip}(1/2)}^{\mathsf{geo}} = \mathsf{EGD}\left(\frac{p}{2(1-c)}, \alpha\right)$. To get a low upper bound, it is best to set $p = 1$. How to choose α and c is not clear because decreasing α will increase $\frac{1}{1-c}$ and vice versa.

To optimize the asymptotics of the tail probabilities $\mathbb{P}[X_1 = n]$ as $n \to \infty$, we want to choose α as small as possible, i.e. very close to $\frac{1}{2}$, accepting that this will make $\frac{1}{1-c} \ge \frac{2\alpha}{2\alpha - 1}$ large. This yields the bound $[\![P]\!](\mathsf{EGD}(1,0)) \preceq_{\mathsf{EGD}} \mathsf{EGD}\left(\left(\frac{1+2\varepsilon}{4\varepsilon}\right), \frac{1}{2}+\varepsilon\right)$, i.e. $\mathbb{P}[X_1 = n] \leq \left(\frac{1}{4\varepsilon}+\frac{1}{2}\right)\left(\frac{1}{2}+\varepsilon\right)^n$, see Zaiser et al. [2024].

To optimize the bound on the expected value of X_1, we want to minimize the bound $\frac{\alpha}{2(1-c)(1-\alpha)^2} \geq \mathbb{E}[X_1]$ from Theorem 4.5. This is achieved under the constraints $0 \leq \alpha, c < 1$ and $\frac{1}{2} \leq \alpha c$ for $\alpha = \frac{\sqrt{5}-1}{2} \approx 0.618$ and $c = \frac{\sqrt{5}+1}{4} \approx 0.809$. At this point, the bound on the expected value is $\mathbb{E}[X_1] \leq \frac{11+5\sqrt{5}}{2} \approx 11.09$. This is quite a bit more than the true value of 1, but it is a finite bound that can be found fully automatically. It can be improved by unrolling the loop a few times before applying the above procedure. This way, our implementation finds much better bounds (see Table 5).

Example 4.11 (Asymmetric random walk)**.** Consider the program representing a biased random walk on $\mathbb{N}$, starting at 1, and stopping when reaching 0:

$$X_1 := 1; X_2 := 0; \textsf{while}\, X_1 > 0 \, \{X_2 \mathrel{+}= 1; \{X_1 \mathrel{+}= 1\} \ [r] \ \{X_1 \mathrel{\dot{-}}= 1\}\}$$

where X_1 is the current position, X_2 is the number of steps taken, and the bias $r < \frac{1}{2}$ is the probability of going right. Denote the loop body by $B := (X_2 \mathrel{+}= 1; \{X_1 \mathrel{+}= 1\} \ [r] \ \{X_1 \mathrel{\dot{-}}= 1\})$. We find

$$\mathsf{EGD}(1,(0,0)) \ [\![X_1 := 1; X_2 := 0]\!]^{\text{geo}} \ \mathsf{EGD}\left(\begin{pmatrix}0\\1\end{pmatrix},(0,0)\right)$$

as the distribution before the loop. Assume a c-contraction invariant $\mathsf{EGD}(\mathbf{P},\boldsymbol{\alpha})$ with $\mathbf{P} \in \mathbb{R}_{\geq 0}^{[2]\times[1]}$ exists. Then one loop iteration transforms it as follows (details in Zaiser et al. [2024]):

$$\mathsf{EGD}\left(\begin{pmatrix}\mathbf{P}_{0,0}\\ \mathbf{P}_{1,0}\end{pmatrix},\boldsymbol{\alpha}\right) \ [\![B]\!]^{\text{geo}} \ \mathsf{EGD}\left(\begin{pmatrix}0 & (1-r)\mathbf{P}_{1,0}\\ 0 & (1-r)\alpha_1\mathbf{P}_{1,0}\\ 0 & (1-r)\alpha_1^2\mathbf{P}_{1,0}+r\mathbf{P}_{1,0}\end{pmatrix},\boldsymbol{\alpha}\right)$$

This yields the following constraints for the loop invariant:

$$\mathsf{EGD}\left(\begin{pmatrix}0\\1\end{pmatrix},(0,0)\right) \preceq_{\mathsf{EGD}} \mathsf{EGD}\left(\begin{pmatrix}\mathbf{P}_{0,0}\\ \mathbf{P}_{1,0}\end{pmatrix},\boldsymbol{\alpha}\right)$$

$$\mathsf{EGD}\left(\begin{pmatrix}0 & (1-r)\mathbf{P}_{1,0}\\ 0 & (1-r)\alpha_1\mathbf{P}_{1,0}\\ 0 & (1-r)\alpha_1^2\mathbf{P}_{1,0}+r\mathbf{P}_{1,0}\end{pmatrix},\boldsymbol{\alpha}\right) \preceq_{\mathsf{EGD}} \mathsf{EGD}\left(\begin{pmatrix}c\mathbf{P}_{0,0}\\ c\mathbf{P}_{1,0}\end{pmatrix},\boldsymbol{\alpha}\right) = \mathsf{EGD}\left(\begin{pmatrix}c\mathbf{P}_{0,0} & c\alpha_2\mathbf{P}_{0,0}\\ c\mathbf{P}_{1,0} & c\alpha_2\mathbf{P}_{1,0}\\ c\alpha_1\mathbf{P}_{1,0} & c\alpha_1\alpha_2\mathbf{P}_{1,0}\end{pmatrix},\boldsymbol{\alpha}\right)$$

which reduce to the following polynomial constraints:

$$0 \leq \mathbf{P}_{0,0} \quad 1 \leq \mathbf{P}_{1,0} \quad (1-r)\mathbf{P}_{1,0} \leq c\alpha_2\mathbf{P}_{0,0} \quad (1-r)\alpha_1\mathbf{P}_{1,0} \leq c\alpha_2\mathbf{P}_{1,0} \quad (1-r)\alpha_1^2\mathbf{P}_{1,0}+r\mathbf{P}_{1,0} \leq c\alpha_1\alpha_2\mathbf{P}_{1,0}$$

besides the obvious ones (every variable is nonnegative and $\alpha_1, \alpha_2, c \in [0,1)$). The projection of the solution set to α_1, α_2 is shown in Fig. 6 for $r = \frac{1}{4}$. The most interesting constraint is the last one, which has the solutions $\alpha_1 \in \left[\frac{c\alpha_2-\sqrt{c^2\alpha_2^2-4r(1-r)}}{2(1-r)}, \frac{c\alpha_2+\sqrt{c^2\alpha_2^2-4r(1-r)}}{2(1-r)}\right]$. It can be shown (see Zaiser et al. [2024]) that such an $\alpha_1 < 1$ exists if and only if $r < \frac{1}{2}$, which makes sense because for $r \geq \frac{1}{2}$, the program has infinite expected running time (see Theorem 4.16). Then all constraints are in fact satisfiable and we get a bound on the distribution of the program:

$$\mathsf{EGD}(1,(0,0)) \ [\![P]\!]^{\text{geo}} \ \mathsf{EGD}\left(\begin{pmatrix}\frac{\mathbf{P}_{0,0}}{1-c}\\ \frac{\mathbf{P}_{1,0}}{1-c}\end{pmatrix},\boldsymbol{\alpha}\right)\Bigg|_{\neg(X_1>0)}^{\text{geo}} = \mathsf{EGD}\left(\begin{pmatrix}\frac{\mathbf{P}_{0,0}}{1-c}\\ 0\end{pmatrix},(0,\alpha_2)\right)$$

The asymptotic bound is $\mathbb{P}[X_2 = n] = \frac{\mathbf{P}_{0,0}}{1-c}\alpha_2^n$, so it's best for α_2 as small as possible. Since $\alpha_2 \geq \frac{\sqrt{4r(1-r)}}{c} > 2\sqrt{r(1-r)}$, the best possible geometric tail bound for $\Pr[X_2 = n]$ is $O((2\sqrt{r(1-r)}+\varepsilon)^n)$.

Interestingly, the exact asymptotic is $\Theta((2\sqrt{r(1-r)})^n \cdot n^{-3/2})$ (see Zaiser et al. [2024]), so the best possible geometric bound is $O((2\sqrt{r(1-r)})^n)$ and our method can get arbitrarily close to it. We could find a bound on $\mathbb{E}[X_2]$ in the same way as in the previous example, but this would be tedious to do manually. Our implementation (again using loop unrolling before applying the above reasoning) finds the bounds shown in Table 5 (for $r = \frac{1}{4}$).

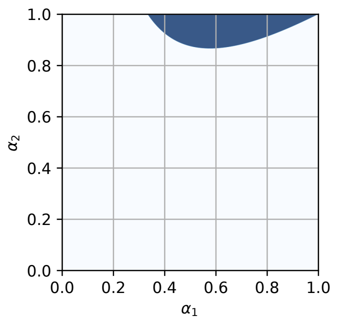


Fig. 6. Solution set for Example 4.11, projected onto α_1, α_2 for $r = \frac{1}{4}$

4.5 Properties

4.5.1 Decidability. For any $\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha})$ occurring in the semantics, each entry of $\mathbf{P}$ and $\boldsymbol{\alpha}$ is a rational function of entries of initial blocks, decay rates, and contraction factors. (In fact, it is linear in initial block entries.) Hence the $\preceq_{\mathsf{EGD}}$ constraints reduce to polynomial inequality constraints if the sizes of the EGDs are known. If we want to find an upper bound, these constraints will contain unknowns, which we have to solve for. Since the existential first-order theory of the reals is decidable, solving these constraints is indeed possible.

Theorem 4.12. *Given a program P and an $\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha})$, it is decidable whether there is an EGD $\mathsf{EGD}(\mathbf{Q}, \boldsymbol{\beta})$ such that $\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha})$ $[\![P]\!]^{\text{geo}}$ $\mathsf{EGD}(\mathbf{Q}, \boldsymbol{\beta})$, assuming the sizes of the intermediate EGDs for joins and contraction invariants in the semantics are fixed.*

This is yet another reason to prefer EGDs over more general classes of distributions: while it may be possible to check $\preceq$ for slightly more complicated classes of distributions with concrete values, solving such a system of $\preceq$-inequalities with unknown values will be much harder. Despite the decidability results, solving our constraints is still not easy because the existential theory of the reals is NP-hard and algorithms only work for small instances in practice.

One might hope that the constraints arising from the geometric bound semantics have nice properties. For example, one might expect the solution set $(\mathbf{Q}, \boldsymbol{\beta})$ to be convex and $\boldsymbol{\beta}$ to be right-closed, i.e. that for any solution $\mathsf{EGD}(\mathbf{Q}, \boldsymbol{\beta})$ and any $\boldsymbol{\gamma} \geq \boldsymbol{\beta}$, there is a solution $\mathsf{EGD}(\mathbf{Q}', \boldsymbol{\gamma})$, because this weakens the bound. However, we found simple counterexamples to both properties (see Zaiser et al. [2024]). We address the problem of practical constraint solving in Section 5.1.

4.5.2 Soundness. We prove that the geometric bound semantics is correct with respect to the standard semantics.

Theorem 4.13 (Soundness). *The rules for the EGD event semantics (Fig. 4) agree with the standard event semantics: $\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha})|_E^{\text{geo}} = \mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha})|_E$. The rules for the EGD statement semantics $[\![-]\!]^{\text{geo}}$ (Fig. 5) are sound: if $\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha})$ $[\![P]\!]^{\text{geo}}$ $\mathsf{EGD}(\mathbf{Q}, \boldsymbol{\beta})$ then $[\![P]\!](\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha})) \preceq \mathsf{EGD}(\mathbf{Q}, \boldsymbol{\beta})$. Furthermore, if $\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha})$ is a probability distribution, then we can bound the normalized distribution:*

$$\frac{[\![P]\!]_{\mathsf{lo}}(\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha}))|_{\mathbb{N}^n}}{\mathsf{EGD}(\mathbf{Q}, \boldsymbol{\beta})(\mathbb{N}^n)} \preceq \operatorname{normalize}([\![P]\!]_{\lightning}(\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha}))) \preceq \frac{\mathsf{EGD}(\mathbf{Q}, \boldsymbol{\beta})}{[\![P]\!]_{\mathsf{lo}}(\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha}))(\mathbb{N}^n)}$$

4.5.3 Loop-free Fragment. It is instructive to look at when $[\![-]\!]^{\text{geo}}$ is precise, i.e. when the right-hand side of the relation is unique and equals the $[\![-]\!]$ semantics. It turns out that this is the case for programs without loops, which is easy to see from the soundness proof.

Theorem 4.14 (Precision for loop-free fragment). *Let P be a loop-free program. Assume that the strict Join^* relation is used in the $[\![-]\!]^{\text{geo}}$ semantics of* if $-$ {$-$} else {$-$}. *Then for all $\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha})$, there is a unique $\mathsf{EGD}(\mathbf{Q}, \boldsymbol{\beta})$ such that $\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha})$ $[\![P]\!]^{\text{geo}}$ $\mathsf{EGD}(\mathbf{Q}, \boldsymbol{\beta})$ and $[\![P]\!](\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha})) = \mathsf{EGD}(\mathbf{Q}, \boldsymbol{\beta})$. Furthermore, each $\beta_i \in \{0, \alpha_i\}$.*

4.5.4 Sufficient Conditions for the Existence of Bounds. When does the constraint problem arising from our semantics actually have a solution? This is hard to characterize in general because the feasible region can be complex, as discussed above. Furthermore, a useful purely syntactic criterion is unlikely to exist because even for the simple asymmetric random walk $\mathsf{while}\, X_1 > 0\, \{\mathsf{if\, flip}(\rho)\; \{X_1 \mathrel{+}= 1\}\, \mathsf{else}\, \{X_1 \mathrel{\dot{-}}= 1\}\}$, a bound can be found if and only if $\rho < \frac{1}{2}$.

Nevertheless, we have succeeded in identifying a sufficient criterion for the existence of an EGD loop bound: roughly speaking, if the difference between new and old values of the variables is the same in each loop iteration, and if there is a linear ranking supermartingale [Chatterjee et al. 2016], then an EGD bound exists. The precise conditions are stated in the following theorem.

Theorem 4.15. *Suppose $L = \mathsf{while}\, E\, \{B\}$ satisfies the following properties:*

1. *B only contains increment and decrement statements ($X_k \mathrel{+}= 1, X_k \mathrel{\dot{-}}= 1$), probabilistic branching ($\mathsf{if\, flip}(\dots)\,\{\dots\}\,\mathsf{else}\,\{\dots\}$), and fail,*
2. *E occurring guarantees that $X_1 \geq a_1 \wedge \cdots \wedge X_n \geq a_n$ where $a_i \in \mathbb{N}$ is the maximum number of decrements of X_i in any program path through B, and*
3. *there is a conical (linear with nonnegative coefficients) combination $\sum_{i=1}^n \lambda_i X_i$ whose expected value decreases after every loop iteration for any initial assignment $\boldsymbol{x}$ that can enter the loop: $\mathbb{E}_{\boldsymbol{X}\sim[\![B]\!](\mathsf{Dirac}(\boldsymbol{x})|_E)}\left[\sum_{i=1}^n \lambda_i X_i\right] < \sum_{i=1}^n \lambda_i x_i \quad$ for all $\boldsymbol{x} \in \mathbb{N}^n$ with $x_i \geq a_i$*

Then, for any initial $\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha})$, there is a solution $(\mathbf{Q}, \boldsymbol{\beta})$ to $\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha})\; [\![L]\!]^{\text{geo}}\; \mathsf{EGD}(\mathbf{Q}, \boldsymbol{\beta})$.

Although the assumptions of this theorem may seem restrictive, they are sufficient to prove the existence of bounds for the random walk (Example 4.11), which is nontrivial to analyze. In that case, $a_1 = 1, a_2 = 0$ and the expected value of X_1 decreases by $1 - 2r > 0$ in each loop iteration.

4.5.5 Necessary Conditions for the Existence of Bounds. We already mentioned that the existence of bounds is not guaranteed in general. An obvious necessary condition is that the true program distribution actually has tails that decay exponentially fast, i.e. can be bounded by an EGD. It turns out that the same must hold for the running time of the program as well.

Theorem 4.16 (Necessary conditions on the running time). *If $\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha})\; [\![P]\!]^{\text{geo}}\; \mathsf{EGD}(\mathbf{Q}, \boldsymbol{\beta})$ for an $\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha})$ then the running time T of P on $\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha})$ can be bounded by an EGD as well. In particular, all its moments $\mathbb{E}[T^k]$ must be finite.*

4.5.6 Loop Unrolling and Convergence. Just like for the residual mass semantics, we would expect the geometric bound semantics to yield tighter and tighter bounds as we unroll loops further and further. It turns out that for the u-fold unrolling $P^{(u)}$ of a program P, we can find upper bounds whose distance from the true distribution decreases exponentially in u.

Theorem 4.17 (Convergence). *Let P be a program containing potentially nested loops and $P^{(u)}$ its u-fold unrolling. Suppose $\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha})\; [\![P]\!]^{\text{geo}}\; \mathsf{EGD}(\mathbf{Q}, \boldsymbol{\beta})$. Then there exist $A \in \mathbb{R}_{\geq 0}, C \in [0, 1)$ and $\mathbf{Q}^{(u)}$ such that $\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha})\; [\![P^{(u)}]\!]^{\text{geo}}\; \mathsf{EGD}\big(\mathbf{Q}^{(u)}, \boldsymbol{\beta}\big) \preceq_{\mathsf{EGD}} \mathsf{EGD}(\mathbf{Q}, \boldsymbol{\beta})$ and*

$$\mathsf{EGD}\Big(\mathbf{Q}^{(u)}, \boldsymbol{\beta}\Big) - [\![P]\!](\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha})) \leq A \cdot C^u \cdot \mathsf{EGD}(\mathbf{Q}, \boldsymbol{\beta})$$

In particular, the distribution bound $\mathsf{EGD}\big(\mathbf{Q}^{(u)}, \boldsymbol{\beta}\big)$ converges in total variation distance to the true distribution $[\![P]\!](\mathsf{EGD}(\mathbf{P}, \boldsymbol{\alpha}))$, as $u \to \infty$. Similarly, the k-th moment bound $\mathbb{E}_{\boldsymbol{X}\sim\mathsf{EGD}(\mathbf{Q}^{(u)},\boldsymbol{\beta})}[X_i^k]$ converges to the true moment $\mathbb{E}_{\boldsymbol{X}\sim[\![P]\!](\mathsf{EGD}(\mathbf{P},\boldsymbol{\alpha}))}[X_i^k]$ for any k.

Note that this theorem does *not* prove that the parameters $\boldsymbol{\beta}$ of the geometric tails converge to the best possible ones. In general, unrolling does not improve the decay rate at all, except for specific cases where the loop is finite and can be fully unrolled. In fact, the derived tail bounds may differ arbitrarily from the true tails, as shown in the next example.

Example 4.18 (Nonoptimal tail bounds)**.** As an example of nonoptimal tail bounds, consider the following program where $\rho \in (0, 1)$:

$$\begin{aligned}&X_1 := 0; X_2 := 0;\\ &\mathsf{while}\,\mathsf{flip}(\rho)\,\{X_1 \mathrel{+}= 1; X_2 \mathrel{+}= 1\};\\ &\mathsf{while}\,X_1 > 0 \land X_2 > 0\,\{X_1 \mathrel{\dot{-}}= 1; X_2 \mathrel{\dot{-}}= 1\}\end{aligned}$$

It is clear that at the end of the program, $X_1 = 0$ and $X_2 = 0$ almost surely. Throughout the program, the same operations are applied to them. However, the geometric bound semantics cannot keep track of this correlation completely because the structure of EGDs can only express correlations in the finite initial block and not in the tails. Hence the best tail bounds we get for X_1 and X_2 are of the form $O((\rho + \delta)^n)$ for any $\delta > 0$, whereas the true tails are zero. So the tail bounds can be arbitrarily far from the true tails. Note, however, that the existence of geometric tails is already useful knowledge: it tells us that the distribution is not heavy-tailed and that all moments are finite.

5 Implementation

We implemented both the residual mass semantics and the geometric bound semantics in a tool called ***Diabolo*** ("**Di**screte **Di**stribution **A**nalysis via **Bo**unds, supporting **L**oops and **O**bservations"). The code is available at github.com/fzaiser/diabolo and on Zenodo [Zaiser 2024a].

Diabolo takes a probabilistic program as input and outputs bounds on probability masses, moments, and tail asymptotics of the posterior distribution of a specified program variable. The output and computational effort can be controlled with various options. The most important one is the loop unrolling limit u, which is common to both semantics. The geometric bound semantics has additional options: the dimension d of the contraction invariants to be synthesized, an optimization objective, as well as the solver and optimizer to be used for the polynomial constraints. The optimization objective is the quantity whose bound should be optimized: the probability mass, the expected value, or the asymptotic decay rates. Depending on which of these is chosen, the best EGD bound can vary significantly (see Section 4.4). Diabolo is implemented in Rust [Matsakis and Klock II 2014] for performance and uses exact rational number computations instead of floating point to ensure the correctness of the results. The distributions in the residual mass semantics are represented as arrays of probability masses. We describe a few practical aspects of the implementation below, most of them pertaining to the geometric bound semantics.

Nondeterminism in the semantics. As pointed out in Section 4.3, there are two places in the geometric bound semantics where choices have to be made: branching and loops. To generate the polynomial constraints arising from while loops, we need to choose the size of the contraction invariants $\mathsf{EGD}(\mathbf{R}, \boldsymbol{\gamma})$. It is usually best to choose the dimensions of $\mathbf{R}$ to be as small as possible (often 1): this reduces both the number of constraints and constraint variables, which facilitates solving. Some programs only have larger contraction invariants, but the dimension rarely needs to be greater than 2. In terms of quality of the bounds, increasing the unrolling limit has a much greater effect (cf. Theorem 4.17). In the semantics of if statements, the choice is in the size of the expansion of the EGDs in the Join relation. In Diabolo, we choose the smallest expansion for speed and memory usage reasons. The size of the EGDs in a loop depends mostly on the size of the contraction invariant, so an increase there automatically reduces the imprecision in Join.

Approximating the support. In the constraint problem arising from a c-contraction invariant $\mathsf{EGD}(\mathbf{R}, \boldsymbol{\gamma})$, the constraint variables corresponding to the entries of $\mathbf{R}$ are "easy" because they only occur linearly, whereas the constraints typically contain higher-degree polynomials in $\boldsymbol{\gamma}$. For this reason, it is desirable to determine the decay rates $\boldsymbol{\gamma}$ in advance, if possible. If the program variable

X_i has finite support $\{a_i, a_i + 1, \ldots, b_i\} \subset \mathbb{N}$, we can set the decay rate γ_i to zero (since the tails of X_i's distribution are zero). But to be able to represent its distribution with $\gamma_i = 0$, we have to choose at least $|\mathbf{R}|_i = b_i + 1$. (In this case, the user-provided dimension d for the contraction invariant is overridden.) Furthermore, we can infer from the support that $\mathbf{R}_{i:j} = \mathbf{0}$ for $j < a_i$. To reap these benefits, we need to analyze the support of the random variables occurring in the program. This is a standard application of abstract interpretation on the interval domain. Overapproximating the support also has the benefit that we automatically know that probabilities outside of it are zero, which can improve the results of both semantics (see Zaiser et al. [2024]).

5.1 Constraint Solving

How can we solve the polynomial constraints arising from the geometric bound semantics? As this is a decidable problem known as the *existential theory of the reals*, we first tried to use SMT solvers, such as Z3 [de Moura and Bjørner 2008] and CVC5 [Barbosa et al. 2022], and the dedicated tool QEPCAD [Brown 2003]. Out of these, Z3 performed the best, but it was only able to solve the simplest problems and scaled badly.

Numerical solvers. Given that even simple programs lead to dozens and sometimes hundreds of constraints, we resorted to numerical solutions instead. The best solver for our purposes turned out to be IPOPT: a library for large-scale nonlinear optimization [Wächter and Biegler 2006]. One issue with IPOPT are rounding errors, so it sometimes returns solutions that are not exactly feasible. To address this, we tighten the constraints by a small margin before handing them over to IPOPT and check the feasibility of the returned solution with exact rational arithmetic. Another disadvantage of IPOPT is that it cannot prove infeasibility – only exact solvers like Z3 and QEPCAD can do so. However, its impressive scalability makes up for these shortcomings.

We also developed a custom solver that transforms the constraint problem into an unconstrained optimization problem (details in Zaiser et al. [2024]) and applies the ADAM optimizer [Kingma and Ba 2015], a popular gradient descent method. We also explored other off-the-shelf solvers for polynomial constraints: IBEX, a rigorous solver based on interval arithmetic, and constrained optimization methods in scipy, the Python scientific computing library; but neither of them performed well. Diabolo includes the solvers IPOPT (the default), our ADAM-based solver, and Z3.

Nested loops. Nested loops lead to cyclic $\leq$-constraints on the decay rates, e.g. $\alpha \leq \beta \land \beta \leq \alpha$. Numerical solvers struggle with such indirect equality constraints, so we added a preprocessing step to detect them and replace the equal variables with a single representative.

Reducing the cost of unrolling. Unrolling loops is essential to obtain tight bounds on posterior masses and moments (it does not affect tail bounds). On the other hand, unrolling increases the complexity of the constraint problem considerably, both in terms of number of variables and constraints. A key observation from Theorem 4.17 can mitigate this problem: the variables that occur nonlinearly, i.e. decay rates and contraction factors, need not be changed as the unrolling count increases. As a consequence, we first solve the constraint problem without unrolling and need only solve a *linear* constraint problem for higher unrolling limits. Since linear programming solvers are much faster and more robust than nonlinear solvers, this approach significantly reduces numerical issues and computation time. In fact, without this technique, several benchmarks in Section 6.2 would not be solvable at all.

5.2 Optimization

Since the geometric bound semantics is nondeterministic, there are many EGD bounds for a given program. Which one is the best depends on what quantity we want to optimize. In Diabolo, the user can specify one of the following optimization objectives to minimize: the bound on the total probability mass, on the expected value, or on the tail asymptotics (i.e. the decay rate).

IPOPT and the ADAM-based solver can also be used for optimization. Moreover, we apply the linear programming solver CBC by the COIN-OR project to optimize the linear variables (keeping the nonlinear ones fixed), which is very fast. By default, Diabolo runs IPOPT, the ADAM-based optimizer, and the linear solver in this order, each improving the solution of the previous one. The ADAM-based optimizer can be slow for larger programs, in which case the user may decide to skip it. However, it often finds better tail bounds than IPOPT, which is why it is included by default.

6 Empirical Evaluation

In this section, we evaluate our two methods in practice to answer the following four questions:

(1) How often is the geometric bound semantics applicable in practice? (Section 6.1)
(2) How tight are the bounds in practice? (Section 6.2)
(3) How do our methods perform compared to previous work? (Section 6.3)
(4) How do our two semantics perform compared to each other? (Section 6.4)

All benchmarks and code to reproduce the results are available [Zaiser 2024a].

6.1 Applicability of the Geometric Bound Semantics

In this section, we empirically investigate the *existence* of geometric bounds, i.e. how often the constraints arising from the geometric bound semantics can be solved. If some bound can be found at all, Theorem 4.17 ensures that it can be made arbitrarily tight by increasing the unrolling limit. (This aspect of the *quality* of bounds is studied in Sections 6.2 and 6.4.) While Theorem 4.15 provides sufficient conditions for existence, we want to test the applicability of the geometric bound semantics in practice in a *systematic* way.

Benchmark selection. For this purpose, we collected benchmarks from the Github repositories of several probabilistic programming languages: Polar [Moosbrugger et al. 2022], Prodigy [Klinkenberg et al. 2024], and PSI [Gehr et al. 2016]. Note that a benchmark being available in a tool's repository does not mean that the tool can solve it. We searched all benchmarks from these repositories for the keyword `while`, in order to find benchmarks with loops. We manually filtered out benchmarks whose loops are actually bounded or that make essential use of continuous distributions (15), negative integers (4), comparisons of two variables (7), or multiplication (1), because our language cannot express these. Some benchmarks using these features could still be translated in other ways to an equivalent program in our language.

We ended up with 43 benchmarks: 9 from Polar, 11 from Prodigy, 9 from PSI, and we added 14 of our own. They include standard probabilistic loop examples (in particular, all examples from this paper, and variations thereof), nested loops, and real-world algorithms, such as probabilistic self-stabilization protocols [Beauquier et al. 1999; Herman 1990; Israeli and Jalfon 1990].

Symbolic inputs. Polar and Prodigy can handle symbolic inputs and symbolic parameters to some extent, which our techniques cannot. One benchmark (`polar/fair_biased_coin`) used symbolic parameters, which we replaced with concrete values. Several benchmarks from Polar and Prodigy have symbolic inputs, i.e. they are parametric in the initial values in $\mathbb{N}$ of the variables. Since our method cannot reason parametrically about the input, we instead put a Geometric$(1/2)$ distribution on such inputs, to cover all possible values. Of course, this yields less information than a method computing a symbolic result that is valid for all input values. But if an EGD bound can be found for this input distribution, then an EGD bound can be found for each possible input value because any Dirac distribution on the input can be bounded by a scaled version of the geometric distribution:

$\mathsf{Dirac}(m) \leq 2^m \cdot \mathsf{Geometric}(1/2)$. Thus, for the question of the existence of bounds, we consider this a valid approach. Note that reasoning about such a geometric distribution on the input is nontrivial (and much harder than reasoning about fixed input values). For instance, Polar cannot always handle it even for benchmarks where it supports the version with symbolic input values.

Methodology. We ran our tool Diabolo for the geometric bound semantics on all benchmarks with a timeout of 5 minutes. The configuration options were mostly left at the defaults for most benchmarks (invariant size: 1, solver: IPOPT). However, we disabled unrolling for all benchmarks since it only affects the quality of bounds, not their existence. For 4 benchmarks, the invariant size had to be increased to 2 or 3 to find bounds. For each benchmark, we recorded the time it took to compute the bounds, or any errors.

Results. Diabolo was able to solve 37 out of the 43 benchmarks (86%); 5 failed because no EGD bound exists (at least 3 of them have infinite expected running time); and 1 timed out due to the complexity of the constraint problem. Among the solved benchmarks, the solution time was at most 3 seconds and typically much less. This demonstrates that our geometric bound semantics is also of practical relevance, in addition to its nice theoretical properties. Detailed results are available in Zaiser et al. [2024], along with statistics about the input program and the constraint problem arising from the geometric bound semantics.

6.2 Quality of the Geometric Bounds

Methodology. Beyond the mere *existence*, we are also interested in the *quality* of the bounds. To this end, we ran Diabolo again on each benchmark where a bound could be found, once optimizing the bound on the expected value, once optimizing the tail bound. The configuration options were left at the default settings with an unrolling limit of 30 for most benchmarks. For expectation bounds, we modified the settings for 23 benchmarks, to adjust the unrolling limit (reported in Table 2), invariant size to 2, 3, or 4 (if smaller values failed or yielded bad bounds), and to skip the ADAM-based optimizer (due to it being slow for some benchmarks). For tail bounds, we set the unrolling limit to 1 and modified the default settings for 15 benchmarks to adjust the invariant size to 2, 3, or 4 and to skip the ADAM-based optimizer (due to it being slow for some benchmarks).

Results. The results are shown in Table 2. We mark with "†" all examples that, to our knowledge, could not be solved automatically before, in the sense of bounding the distribution's moments and tails without user intervention. Note that no other existing tool is able to obtain *exponential* tail bounds like ours on any benchmark (unless the tails are zero). One can see that all bounds are nontrivial and the upper and lower bounds on the expected value are usually close together. However, it seems to be harder to find good bounds for benchmarks where the tails of the distribution decay slowly, such as the asymmetric random walks (ours/*-asym-rw). Most of the tail bounds are also very close to the theoretical optimum where the exact tail bound could be manually determined. This demonstrates that our geometric bound semantics generally yields useful bounds.

6.3 Comparison with Previous Work

Comparison with GuBPI. Our residual mass semantics shares many characteristics with GuBPI (see Table 1). However, GuBPI is designed for a more general setting with continuous sampling and soft conditioning. As a consequence, when applied to discrete probabilistic programs with only hard conditioning, there is a lot of overhead. To demonstrate this, we ran both tools on Examples 1.1, 4.10 and 4.11, configured to produce the same bounds. The results (Table 3) show that Diabolo's residual mass semantics is several orders of magnitude faster than GuBPI.

Table 2. Diabolo's bounds on the expected value and the tail asymptotics for the benchmarks from Section 6.1 where bounds exist. Results are rounded to 4 significant digits. (#U: unrolling limit; EV: expected value; "?": ground truth could not be determined; "†": could not be solved by an automatic tool before, in the sense of bounding the distribution's moments and tails without user intervention; timeout: 5 minutes exceeded.)

Benchmark	#U	True EV	EV bound	Time	True tail	Tail bound	Time
polar/c4B_t303 †	30	0.1787...	[0.1787, 0.1788]	0.24 s	?	$O(0.5001^n)$	0.12 s
polar/coupon_collector2	30	2	[1.999, 2.001]	0.16 s	$\Theta(0.5^n)$	$O(0.5071^n)$	0.11 s
polar/fair_biased_coin	30	0.5	[0.4999, 0.5001]	0.03 s	0	0	0.02 s
polar/las_vegas_search	200	20	[19.98, 26.79]	2.29 s	$\Theta(0.9523\ldots^n)$	$O(0.9524^n)$	0.72 s
polar/linear01 †	30	0.375	[0.3749, 0.3751]	0.03 s	0	0	0.03 s
polar/simple_loop	30	1.3	[1.299, 1.301]	0.02 s	0	0	0.02 s
prodigy/bit_flip_conditioning	30	1.254...	[1.254, 1.255]	0.28 s	?	$O(0.6254^n)$	0.19 s
prodigy/brp_obs †	30	4.989...e-10	[4.989e-10, 1.489e-09]	0.61 s	0	0	3.83 s
prodigy/condand †	30	0.75	[0.7499, 0.7501]	0.15 s	?	$O(0.5001^n)$	0.07 s
prodigy/dep_bern †	30	0.5	[0.4999, 0.5124]	1.00 s	$\Theta(0.3333\ldots^n)$	$O(0.34^n)$	0.17 s
prodigy/endless_conditioning	30	undef	undef	0.15 s	0	0	0.13 s
prodigy/geometric	30	2	[1.999, 2.007]	1.09 s	$\Theta(0.5^n)$	$O(0.5066^n)$	0.60 s
prodigy/ky_die	30	3.5	[3.499, 3.501]	0.18 s	0	0	0.08 s
prodigy/n_geometric †	30	1	[0.998, 1.068]	0.08 s	$\Theta(0.6666\ldots^n)$	$O(0.673^n)$	0.07 s
prodigy/trivial_iid †	30	3.5	[3.499, 3.503]	2.60 s	$\Theta(0.8318\ldots^n)$	$O(0.836^n)$	0.16 s
psi/beauquier-etal3	30	?	✗ (timeout)	t/o	?	$O(0.7952^n)$	17.64 s
psi/cav-example7	80	10.41...	[10.41, 10.51]	1.71 s	?	$O(0.8927^n)$	0.12 s
psi/dieCond (Ex. 1.1) †	40	1.5	[1.499, 1.501]	0.15 s	$\Theta(0.3333\ldots^n)$	$O(0.3395^n)$	0.13 s
psi/ex3	30	0.6666...	[0.6666, 0.6667]	0.04 s	0	0	0.06 s
psi/ex4	30	0.6666...	[0.6666, 0.6667]	0.23 s	0	0	0.27 s
psi/fourcards	30	0.2642...	[0.264, 0.2648]	0.46 s	0	0	0.34 s
psi/herman3	30	1.333...	[1.333, 1.334]	62.08 s	?	$O(0.5002^n)$	0.48 s
psi/israeli-jalfon3	30	0.6666...	[0.6666, 0.6668]	1.71 s	?	$O(0.2501^n)$	0.15 s
psi/israeli-jalfon5	30	?	✗ (timeout)	t/o	?	$O(0.6583^n)$	7.13 s
ours/1d-asym-rw (Ex. 4.11) †	70	2	[1.999, 2.542]	1.47 s	$\Theta(\frac{0.8660\ldots^n}{n^{1.5}})$	$O(0.8682^n)$	0.12 s
ours/2d-asym-rw †	70	?	[7.394, 100.4]	79.60 s	?	$O(0.9385^n)$	4.88 s
ours/3d-asym-rw †	30	?	[9.443, 6.85e+05]	289.94 s	?	$O(0.9812^n)$	6.35 s
ours/asym-rw-conditioning †	70	2.444...	[2.316, 2.588]	4.85 s	0	0	0.05 s
ours/coupon-collector5	80	11.41...	[11.41, 11.56]	47.93 s	$\Theta(0.8^n)$	$O(0.8002^n)$	15.56 s
ours/double-geo †	30	2	[1.999, 2.01]	0.32 s	$\Theta(0.7071\ldots^n)$	$O(0.711^n)$	0.12 s
ours/geometric (Ex. 4.10) †	30	1	[0.9999, 1.006]	0.12 s	$\Theta(0.5^n)$	$O(0.5037^n)$	0.10 s
ours/grid †	30	2.75	[2.749, 2.766]	2.39 s	?	$O(0.5285^n)$	0.30 s
ours/imprecise_tails (Ex. 4.18) †	30	0	[0, 0.007054]	2.36 s	0	$O(0.5001^n)$	0.18 s
ours/israeli-jalfon4	30	?	[1.457, 1.458]	121.04 s	?	$O(0.5011^n)$	0.44 s
ours/nested †	5	?	[0.9152, 10.57]	12.59 s	?	$O(0.4998^n)$	0.40 s
ours/sub-geom †	30	0.6666...	[0.6666, 0.6667]	0.06 s	?	$O(0.5001^n)$	0.10 s
ours/sum-geos	80	8	[7.998, 8.001]	2.91 s	$\Theta(0.875^n)$	$O(0.8751^n)$	0.30 s

Table 3. Comparison of the running time of GuBPI [Beutner et al. 2022] and our residual mass semantics (as implemented in Diabolo) to produce the same bounds

Benchmark	GuBPI	Residual mass semantics	Speedup factor
Simple counter (Example 4.10)	0.7 s	0.006 s	$1.2 \cdot 10^2$
Asymmetric random walk (Example 4.11)	90 s	0.002 s	$4.5 \cdot 10^4$
Mossel's die paradox (Example 1.1)	156 s	0.0008 s	$2 \cdot 10^5$

Wang et al. [2024] subsequently improved on GuBPI's results, but we were unable to compare with their system because they use proprietary software. However, they report performance improvements of at most a factor of 15 over GuBPI, which is insufficient to bridge the factor of over 100 for Example 4.10 between GuBPI and our residual mass semantics, let alone the factor of over 10^5 for Example 1.1. This demonstrates that the residual mass semantics is more effective than existing tools for discrete probabilistic programs.

Comparison with Polar. We compare the geometric bound semantics (as implemented in Diabolo) with Polar [Moosbrugger et al. 2022], the only other tool that can be used to bound moments. In fact,

Table 4. Comparison of Diabolo with Polar [Moosbrugger et al. 2022] on the benchmarks from Table 2 where Polar can compute the expected value (EV). Diabolo's results are the same as in Table 2, but listed again for an easier comparison. (t/o: timeout of 5 minutes exceeded.)

Benchmark	Polar		Diabolo (ours)	
	Exact EV	Time	EV bound	Time
polar/coupon_collector2	2	0.26 s	[1.999, 2.001]	0.16 s
polar/fair_biased_coin	1/2	0.41 s	[0.4999, 0.5001]	0.03 s
polar/las_vegas_search	✗	t/o	[19.98, 26.79]	2.29 s
polar/simple_loop	13/10	0.20 s	[1.299, 1.301]	0.02 s
prodigy/geometric	2	0.46 s	[1.999, 2.007]	1.09 s
prodigy/ky_die	✗	t/o	[3.499, 3.501]	0.18 s
psi/beauquier-etal3	✗	t/o	✗	t/o
psi/cav-example7	✗	t/o	[10.41, 10.51]	1.71 s
psi/ex3	2/3	0.58 s	[0.6666, 0.6667]	0.04 s
psi/ex4	2/3	0.21 s	[0.6666, 0.6667]	0.23 s
psi/fourcards	✗	t/o	[0.264, 0.2648]	0.46 s
psi/herman3	4/3	19.53 s	[1.333, 1.334]	62.08 s
psi/israeli-jalfon3	2/3	19.40 s	[0.6666, 0.6668]	1.71 s
psi/israeli-jalfon5	✗	t/o	✗	t/o
ours/coupon-collector5	137/12	74.86 s	[11.41, 11.56]	47.93 s
ours/geometric	1	0.59 s	[0.9999, 1.006]	0.12 s
ours/israeli-jalfon4	✗	t/o	[1.457, 1.458]	121.04 s
ours/sum-geos	8	0.19 s	[7.998, 8.001]	2.91 s

Polar can compute moments exactly, but does not bound the tail asymptotics.[5] We only included benchmarks from Table 2 where Polar can compute the expected value (EV), at least in theory. (We had to extend Polar slightly to support a geometric distribution as the initial distribution.) Polar was run with a timeout of 5 minutes, like Diabolo. The results (Table 4) demonstrate that Diabolo is often faster, and sometimes much faster, than Polar and applicable to more benchmarks.

Comparison of tail bounds. There are several existing program analyses that can bound the tail distribution of the running time of a probabilistic program (see Section 7), but only one [Chatterjee et al. 2016] achieves exponential bounds $O(c^n)$ for $c < 1$ like our bounds. This is particularly interesting as their assumptions (bounded differences and the existence of a linear ranking supermartingale) are remarkably similar to the conditions of our Theorem 4.15. Since the code for the experiments by Chatterjee et al. [2016] is not available, we do a manual comparison on Examples 4.10 and 4.11. Our geometric bound semantics yields the bounds $O((\frac{1}{2} + \varepsilon)^n)$ and $O((2\sqrt{r(1-r)} + \varepsilon)^n)$, respectively, whereas their method yields $O((\exp(-1/2))^n)$ and $O((\sqrt{\exp(-(1-2r)^2)})^n)$, where r is the bias of the random walk. It is not hard to see that our bounds are tighter (details in Zaiser et al. [2024]).

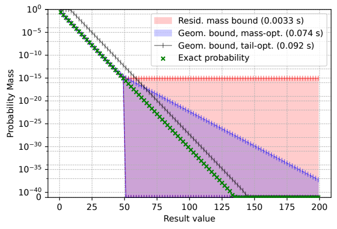
6.4 Comparison Between Our Two Semantics

We compare the *quality* of the bounds from the two semantics on 5 examples: the simple counter (Example 4.10), the asymmetric random walk (Example 4.11), the introductory example (Example 1.1), the coupon collector problem with 5 coupons (ours/coupon-collector5), and Herman's self-stabilization protocol with 3 processes (adapted from psi/herman3). They were run with a loop unrolling limit of 50, 70, 40, 80, and 30 respectively, except for the geometric tail bounds where the limit was set to 1. We report bounds on the first two (raw) moments and the tails (Table 5). Note that the residual mass semantics (not just the lower bound semantics) is needed for lower bounds

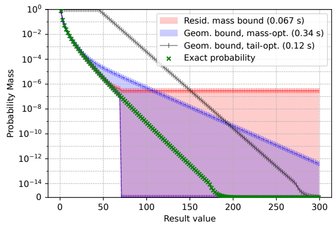
[5]One could derive the tail asymptotic bound $O(n^{-k})$ from the k-th moment via Markov's inequality. But this asymptotic bound is very weak as it applies to any distribution with a finite k-th moment.

Table 5. Comparison of the quality of moment and tail bounds for various examples between the residual mass and geometric bound semantics

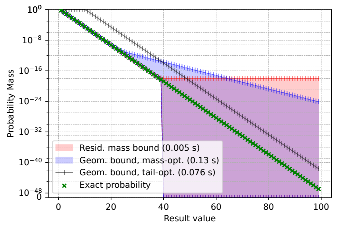
Example	Method	Expected value	2nd moment	Tail	Time
Simple counter (Example 4.10)	exact	1	3	$\Theta(0.5^n)$	
	res. mass sem.	$\geq 1 - 5 \cdot 10^{-14}$	$\geq 3 - 3 \cdot 10^{-12}$	n/a	0.003 s
	geom. bound sem.	≤ 1.000037	≤ 3.00017	$O(0.5037^n)$	0.092 s
Random walk (Example 4.11)	exact	2	10	$\Theta(n^{-3/2} 0.8660\ldots^n)$	
	res. mass sem.	≥ 1.99997	≥ 9.998	n/a	0.067 s
	geom. bound sem.	≤ 2.55	≤ 21.8	$O(0.8682^n)$	1.47 s
Introductory "Die paradox" (Example 1.1)	exact	1.5	3	$\Theta(0.3333\ldots^n)$	
	res. mass sem.	$\geq 1.5 - 2 \cdot 10^{-16}$	$\geq 3 - 4 \cdot 10^{-16}$	n/a	0.005 s
	geom. bound sem.	$\leq 1.5 + 8 \cdot 10^{-7}$	$\leq 3 + 3 \cdot 10^{-6}$	$O(0.3395^n)$	0.097 s
Coupon collector (5 coupons)	exact	11.41666...	155.513888...	$\Theta(0.8^n)$	
	res. mass sem.	≥ 11.41665	≥ 155.5131	n/a	0.88 s
	geom. bound sem.	≤ 11.56	≤ 158.5	$O(0.8002^n)$	22.6 s
Herman's self-stabilization (3 processes)	exact	1.333...	4.222...	?	
	res. mass sem.	≥ 1.3333332	≥ 4.222220	n/a	0.509 s
	geom. bound sem.	≤ 1.3339	≤ 4.226	$O(0.5002^n)$	34.8 s



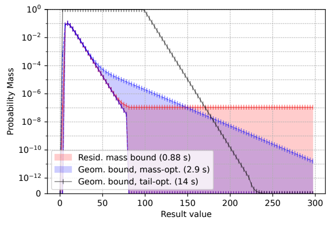
(a) Geom. counter (Example 4.10)



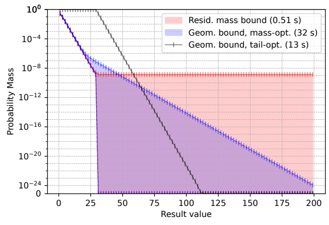
(b) Asym. rand. walk (Example 4.11)



(c) Die paradox (Example 1.1)



(d) Coupon collector problem with 5 coupons



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

Fig. 7. Comparison of the residual mass and geometric bound semantics. Note that the probability masses (y-axis) are on a logarithmic scale, except for the lowest part, which is linear so as to include 0.

on moments because they require upper bounds on the normalizing constant. The optimization objective for the EGD bounds was the expected value for the moment bounds and the decay rate for the tail bound. The residual mass semantics is faster and finds good lower bounds on the moments, but cannot find upper bounds or tail bounds. The geometric bound semantics can find upper bounds on both, but takes much more time than the residual mass semantics.

The bounds on the probability masses can be found in Fig. 7. We report EGD bounds for two optimization objectives: the total probability mass ("mass-optimized") and the decay rate ("tail-optimized"). The residual mass semantics is faster and yields tighter bounds for the probability masses on small values, but the bound is flat, i.e. the difference between upper and lower bounds is constant. In contrast, the geometric bound semantics finds upper bounds that keep decreasing like a geometric distribution even for large numbers beyond the unrolling limit. On the other hand, its upper bound is somewhat worse for small values than the residual mass semantics. This is because the geometric bounds arise from the contraction invariant, which requires a uniform decrease of the distribution. So the geometric bounds need to lose precision either for large values (when minimizing the total mass bound), or small values (when minimizing the tail bounds). In fact, the tail-optimized bound is much worse for small values, but eventually becomes almost parallel to the exact solution, since their asymptotics are almost equal up to a constant factor.

7 Related Work

A summary of the most relevant work is given in Table 1: guaranteed bounds [Beutner et al. 2022; Wang et al. 2024], exact Bayesian inference with loops [Klinkenberg et al. 2024], and exact loop analysis with moments [Moosbrugger et al. 2022]. This section presents a more detailed account.

Guaranteed bounds. The two most directly related pieces of work are Beutner et al. [2022] and Wang et al. [2024]. Beutner et al. [2022] compute guaranteed bounds on posterior probabilities: they partition the trace space (the space of all sampled values during program execution) into boxes or, more generally, polytopes, in such a way that they can get upper and lower bounds on the likelihood in each partition. They also present an interval type system to overapproximate recursion. For discrete probabilistic programs, the effective bounds are comparable to our residual mass semantics (Section 6.3), but their computation is slower and they are not proven to converge.

Wang et al. [2024] introduce a new approach to bounding fixed points and can even find nontrivial bounds for "score-recursive" programs, i.e. programs with loops where the likelihood can increase in each iteration, and may be unbounded.

Both methods focus on continuous distributions and obtain bounds on the posterior probability of the result X of a program being in a certain interval, i.e. bounds of the form $\mathbb{P}[X \in [a,b]] \in [p,q] \subseteq [0,1]$. If X has infinite support, this is not enough to bound the moments or tail probability asymptotics. Our geometric bound semantics bounds the *whole distribution* and therefore does not suffer from these restrictions. On the other hand, our programming language is more restricted and we do not support continuous distributions.

Verified samplers. Another approach to ensuring correctness of approximate inference is to verify the sampler itself. The Zar system compiles discrete probabilistic programs with loops to provably correct samplers that can be used to form simple Monte Carlo estimates of conditional probability masses and moments and is fully verified in the Coq proof assistant [Bagnall et al. 2023].

Exact Bayesian inference. Exact inference is intractable in general because it requires analytical solutions to infinite sums or integrals [Gehr et al. 2016]. Thus exact inference systems either have to restrict programs to a tractable subclass or may fail on some inputs. In the former category are Dice [Holtzen et al. 2020], which only supports finite discrete distributions, and SPPL [Saad et al. 2021], which supports continuous distributions, but imposes restrictions on their use, and Genfer [Zaiser et al. 2023], which allows some infinite-support distributions but restricts operations on variables. In the latter category are the systems PSI [Gehr et al. 2016] and Hakaru [Narayanan et al. 2016], which rely on computer algebra to find a closed-form solution for the posterior.

We are aware of only one piece of work that tackles exact inference for loops [Klinkenberg et al. 2024]. They build on the idea to use generating functions as an exact representation of distributions with infinite support [Klinkenberg et al. 2020]. Zaiser et al. [2023] combine this idea with automatic differentiation to perform Bayesian inference on loop-free probabilistic programs and can even support continuous sampling (but only discrete observations). Back in the fully discrete setting, Klinkenberg et al. [2024] can perform exact Bayesian inference on a *single probabilistic loop without nesting*, but only if a loop invariant template is provided, which specifies the shape of the loop invariant, but may contain holes that have to be filled with real numbers. They can synthesize these numbers and verify the resulting invariant, from which it is easy to obtain the exact posterior distribution, parameterized in the program inputs. However, the hard problem of finding the right shape of the loop invariant falls on the user. In fact, even if a loop invariant exists, it cannot always be specified in their language [Klinkenberg et al. 2024, Example 25]. In contrast, our method is fully automatic, so it does not suffer from these issues. As a trade-off, our method is not exact and only works on fixed program inputs.

Exact loop analysis with moments. There has been some work on studying probabilistic loop behavior by synthesizing invariants for the moments of the random variables. Bartocci et al. [2019] consider Prob-solvable loops, which do not have a stopping condition and whose body consists of polynomial assignments with random coefficients. They obtain invariants for the moments of the program variables by solving a system of recurrence equations arising from the fixed structure of the loop body. Amrollahi et al. [2022] extend this method to a larger class of loops, with fewer restrictions on variable dependencies. Moosbrugger et al. [2022] support (restricted) if-statements and state-dependent distribution parameters. They use the moments to bound tail probabilities and can reconstruct the probabilities masses of the distribution if it has finite support. Kofnov et al. [2024] allow Prob-solvable loops with exponential and trigonometric updates in the loop body, by leveraging the characteristic function.

Our approach differs from the above-mentioned pieces of work in several regards. On the one hand, these approaches yield exact moments and can handle continuous distributions and more primitive operations (e.g. multiplication). On the other hand, they do not support branching on variables with infinite support, nested loops, or conditioning. Our method can handle these constructs, and bounds the whole distribution, not just the moments. As a consequence, we get bounds on probability masses and exponential bounds on the tails, which are better than the polynomial bounds that can be derived from the moments in previous works.

Cost analysis & martingales. Martingales are a common technique for cost analysis of probabilistic programs, i.e. computing bounds on the total cost of a program. The cost can be running time, resource usage, or a utility/reward. A martingale is a sequence of random variables $X_0, X_1, \dots$ such that $\mathbb{E}[X_i] < \infty$ and $\mathbb{E}[X_{i+1} \mid X_1, \dots, X_i] = X_i$ for all $i \in \mathbb{N}$. A supermartingale only requires $\mathbb{E}[X_{i+1} \mid X_1, \dots, X_i] \leq X_i$. A ranking supermartingale (RSM) requires $\mathbb{E}[X_{i+1} \mid X_1, \dots, X_i] \leq X_i - \varepsilon \cdot [X_n > 0]$ for some $\varepsilon > 0$. (There are small variations in the definitions.)

Chakarov and Sankaranarayanan [2013] introduce the concept of a RSM to prove almost sure termination of probabilistic programs. Chatterjee et al. [2016] extend RSMs to probabilistic programs with nondeterminism and use them to derive expectation and tail bounds on the running time T. Then the program's expected termination time is $\mathbb{E}[T] \leq \frac{\mathbb{E}(X_0)}{\varepsilon}$. Assuming the martingale is difference-bounded (i.e. $|X_{i+1} - X_i| < C$ for all i), they derive exponential tail bounds on the running time ($\mathbb{P}[T = n] = O(c^n)$) from Azuma's inequality for supermartingales.

Kura et al. [2019] also deal with termination probabilities, but extend the method to higher moments $\mathbb{E}[T^k]$ of the running time. Via Markov's inequality, they obtain polynomial tail bounds on the termination time ($\mathbb{P}[T = n] = O(n^{-k})$), which are asymptotically weaker than Chatterjee et al. [2016]'s exponential bounds, but more generally applicable (no need for bounded differences). Wang et al. [2021] improve on this by bounding *central* moments, from which they obtain better (but still polynomial) tail bounds than Kura et al. [2019]. In subsequent work by Ngo et al. [2018], Wang et al. [2019], and Chatterjee et al. [2024], the focus shifts to more general costs than running time and relaxing requirements around bounded updates and nonnegativity of costs.

Martingale-based methods can handle continuous distributions and more primitive program operations (e.g. multiplication). Their bounds are also typically parameterized in the program inputs, whereas our bounds only apply to fixed inputs. Conversely, our method yields bounds not just on one program variable (e.g. the running time), but on the distribution of all variables; we support conditioning; and our bounds can be made arbitrarily tight (Theorems 3.5 and 4.17).

8 Conclusion

We advance the concept of guaranteed bounds for discrete probabilistic programs. Such bounds are both more automatable and generally applicable than exact methods, as well as providing more guarantees than sampling-based methods. Our residual mass semantics finds flat tail bounds, improving upon the state of the art in terms of simplicity, speed, and provable guarantees. Our geometric bound semantics adds two novel concepts (contraction invariants and EGDs) to the toolbox of automated probabilistic program analysis, which currently relies predominantly on martingales, moment analysis, and generating functions. Since this semantics overapproximates the *whole distribution* (as opposed to individual probabilities or moments), it can bound probability masses, moments, and tails at the same time. Both semantics have desirable theoretical properties, such as soundness and convergence, and our empirical studies demonstrate their applicability and effectiveness on a variety of benchmarks.

8.1 Future Work

It should be possible to handle (demonic) nondeterminism, as this corresponds to finding a bound ν on the maximum of two EGD bounds μ_1, μ_2, which can be specified using inequalities $\mu_1 \preceq \nu$ and $\mu_2 \preceq \nu$. One could also try to generalize the shape of upper bounds from EGDs to a more expressive subclass of discrete phase-type distributions. Furthermore, it would be interesting to illuminate the theoretical connections with martingale analysis and the quality of the derivable bounds, which was investigated empirically in Section 6.3.

Negative numbers. Our language is restricted to variables in $\mathbb{N}$. The residual mass semantics can easily be extended to $\mathbb{Z}$ since it only requires an exact semantics for the loop-free fragment, which is straightforward for distributions with finite support. We believe that the geometric bound semantics can also be extended to $\mathbb{Z}$, but this is more involved. Even for the simple case of a single variable, one EGD is not enough, but we need two: one for the negative part and one for the nonnegative part. In the case of n variables, we need one $\mathsf{EGD}(\mathbf{P}^{\boldsymbol{o}}, \boldsymbol{\alpha}^{\boldsymbol{o}})$ for each orthant $\boldsymbol{o} \in \{+, -\}^n$. We do not foresee any serious conceptual difficulties extending the geometric bound semantics to operate on such a family of orthant EGDs, but the notation and proofs become quite cumbersome because $\mathbf{P}$ takes *two* multi-indices, not just one. In the interest of clarity and conceptual simplicity, we decided not to pursue this idea further.

Continuous distributions. Variables with values in $\mathbb{R}$ that can be sampled from continuous distributions are much harder to support. The residual mass semantics can build on the exact semantics for the loop-free fragment by Zaiser et al. [2023] to achieve partial support for continuous distributions and variables in $\mathbb{R}_{\geq 0}$ [Zaiser 2024b, Chapter 5]. For the geometric bound semantics, it may be possible to support continuous distributions by extending the exponential distribution to "eventually exponential distributions" (analogously to EGDs extending geometric distributions) which would consist of an initial block using, e.g. a polynomial bound on the probability density function on a compact interval $[-a, a]$ of $\mathbb{R}$, and exponential distributions on $(-\infty, -a]$ and $[a, \infty)$. Whether and how this could work in detail is unclear and requires further investigation.

Data Availability Statement

The artifact for this paper (consisting of the Diabolo tool and benchmarks) is archived on Zenodo [Zaiser 2024a]. The latest version is available on Github: https://github.com/fzaiser/diabolo.

Acknowledgments

We would like to thank Maria Craciun, Eva Darulova, Joost-Pieter Katoen, Lutz Klinkenberg, Sam Staton, Dominik Wagner, Peixin Wang, Đorđe Žikelić, and the anonymous reviewers for their helpful comments and discussions.

This research was supported by the Engineering and Physical Sciences Research Council (studentship 2285273, grant EP/T006579) and the National Research Foundation, Singapore, under its RSS Scheme (NRF-RSS2022-009).

References

Daneshvar Amrollahi, Ezio Bartocci, George Kenison, Laura Kovács, Marcel Moosbrugger, and Miroslav Stankovic. 2022. Solving Invariant Generation for Unsolvable Loops. In *SAS 2022: Static Analysis - 29th International Symposium, Auckland, New Zealand, December 5-7, 2022, Proceedings (Lecture Notes in Computer Science, Vol. 13790)*. Springer, 19–43. https://doi.org/10.1007/978-3-031-22308-2_3

Alexander Bagnall, Gordon Stewart, and Anindya Banerjee. 2023. Formally Verified Samplers from Probabilistic Programs with Loops and Conditioning. *Proc. ACM Program. Lang.* 7, PLDI (2023), 1–24. https://doi.org/10.1145/3591220

Haniel Barbosa, Clark W. Barrett, Martin Brain, Gereon Kremer, Hanna Lachnitt, Makai Mann, Abdalrhman Mohamed, Mudathir Mohamed, Aina Niemetz, Andres Nötzli, Alex Ozdemir, Mathias Preiner, Andrew Reynolds, Ying Sheng, Cesare Tinelli, and Yoni Zohar. 2022. cvc5: A Versatile and Industrial-Strength SMT Solver. In *TACAS 2022: Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 13243)*. Springer, 415–442. https://doi.org/10.1007/978-3-030-99524-9_24

Gilles Barthe, Joost-Pieter Katoen, and Alexandra Silva. 2020. *Foundations of Probabilistic Programming*. Cambridge University Press.

Ezio Bartocci, Laura Kovács, and Miroslav Stankovic. 2019. Automatic Generation of Moment-Based Invariants for Prob-Solvable Loops. In *ATVA 2019: Automated Technology for Verification and Analysis - 17th International Symposium, Taipei, Taiwan, October 28-31, 2019, Proceedings (Lecture Notes in Computer Science, Vol. 11781)*. Springer, 255–276. https://doi.org/10.1007/978-3-030-31784-3_15

Joffroy Beauquier, Maria Gradinariu, and Colette Johnen. 1999. Memory Space Requirements for Self-Stabilizing Leader Election Protocols. In *PODC 1999: Proceedings of the Eighteenth Annual ACM Symposium on Principles of Distributed Computing, Atlanta, Georgia, USA, May 3-6, 1999*. ACM, 199–207. https://doi.org/10.1145/301308.301358

Raven Beutner, C.-H. Luke Ong, and Fabian Zaiser. 2022. Guaranteed bounds for posterior inference in universal probabilistic programming. In *PLDI 2022: 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation, San Diego, CA, USA, June 13-17, 2022*. ACM, 536–551. https://doi.org/10.1145/3519939.3523721

Mogens Bladt and Bo Friis Nielsen. 2017. *Matrix-Exponential Distributions in Applied Probability*. Springer US. https://doi.org/10.1007/978-1-4939-7049-0

Christopher W. Brown. 2003. QEPCAD B: a program for computing with semi-algebraic sets using CADs. *SIGSAM Bull.* 37, 4 (2003), 97–108. https://doi.org/10.1145/968708.968710

Azucena Campillo Navarro. 2018. *Order statistics and multivariate discrete phase-type distributions*. Ph. D. Dissertation. DTU Compute.

Aleksandar Chakarov and Sriram Sankaranarayanan. 2013. Probabilistic Program Analysis with Martingales. In *CAV 2013: Computer Aided Verification - 25th International Conference, Saint Petersburg, Russia, July 13-19, 2013. Proceedings (Lecture Notes in Computer Science, Vol. 8044)*. Springer, 511–526. https://doi.org/10.1007/978-3-642-39799-8_34

Krishnendu Chatterjee, Hongfei Fu, Petr Novotný, and Rouzbeh Hasheminezhad. 2016. Algorithmic analysis of qualitative and quantitative termination problems for affine probabilistic programs. In *POPL 2016: Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, St. Petersburg, FL, USA, January 20-22, 2016*. ACM, 327–342. https://doi.org/10.1145/2837614.2837639

Krishnendu Chatterjee, Amir Kafshdar Goharshady, Tobias Meggendorfer, and Đorđe Žikelić. 2024. Quantitative Bounds on Resource Usage of Probabilistic Programs. *Proceedings of the ACM on Programming Languages* 8, OOPSLA1 (2024), 362–391. https://doi.org/10.1145/3649824

Paul Dagum and Michael Luby. 1993. Approximating Probabilistic Inference in Bayesian Belief Networks is NP-Hard. *Artif. Intell.* 60, 1 (1993), 141–153. https://doi.org/10.1016/0004-3702(93)90036-B

Leonardo Mendonça de Moura and Nikolaj S. Bjørner. 2008. Z3: An Efficient SMT Solver. In *TACAS 2008: Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings (Lecture Notes in Computer Science, Vol. 4963).* Springer, 337–340. https://doi.org/10.1007/978-3-540-78800-3_24

Timon Gehr, Sasa Misailovic, and Martin T. Vechev. 2016. PSI: Exact Symbolic Inference for Probabilistic Programs. In *CAV 2016: Computer Aided Verification - 28th International Conference, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 9779).* Springer, 62–83. https://doi.org/10.1007/978-3-319-41528-4_4

Andrew Gelman, John B. Carlin, Hal S. Stern, David B. Dunson, Aki Vehtari, and Donald B. Rubin. 2013. *Bayesian Data Analysis.* Chapman and Hall/CRC. https://doi.org/10.1201/b16018

Ted Herman. 1990. Probabilistic Self-Stabilization. *Inf. Process. Lett.* 35, 2 (1990), 63–67. https://doi.org/10.1016/0020-0190(90)90107-9

Steven Holtzen, Guy Van den Broeck, and Todd D. Millstein. 2020. Scaling exact inference for discrete probabilistic programs. *Proc. ACM Program. Lang.* 4, OOPSLA (2020), 140:1–140:31. https://doi.org/10.1145/3428208

Amos Israeli and Marc Jalfon. 1990. Token Management Schemes and Random Walks Yield Self-Stabilizing Mutual Exclusion. In *Proceedings of the Ninth Annual ACM Symposium on Principles of Distributed Computing, Quebec City, Quebec, Canada, August 22-24, 1990.* ACM, 119–131. https://doi.org/10.1145/93385.93409

Nils Jansen, Christian Dehnert, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Lukas Westhofen. 2016. Bounded Model Checking for Probabilistic Programs. In *ATVA 2016: Automated Technology for Verification and Analysis - 14th International Symposium, Chiba, Japan, October 17-20, 2016, Proceedings (Lecture Notes in Computer Science, Vol. 9938).* 68–85. https://doi.org/10.1007/978-3-319-46520-3_5

Benjamin Lucien Kaminski and Joost-Pieter Katoen. 2015. On the Hardness of Almost-Sure Termination. In *MFCS 2015: Mathematical Foundations of Computer Science 2015 - 40th International Symposium, Milan, Italy, August 24-28, 2015, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 9234).* Springer, 307–318. https://doi.org/10.1007/978-3-662-48057-1_24

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *ICLR 2015: 3rd International Conference on Learning Representations, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings.* http://arxiv.org/abs/1412.6980

Lutz Klinkenberg, Kevin Batz, Benjamin Lucien Kaminski, Joost-Pieter Katoen, Joshua Moerman, and Tobias Winkler. 2020. Generating Functions for Probabilistic Programs. In *LOPSTR 2020: Logic-Based Program Synthesis and Transformation - 30th International Symposium, Bologna, Italy, September 7-9, 2020, Proceedings (Lecture Notes in Computer Science, Vol. 12561).* Springer, 231–248. https://doi.org/10.1007/978-3-030-68446-4_12

Lutz Klinkenberg, Christian Blumenthal, Mingshuai Chen, Darion Haase, and Joost-Pieter Katoen. 2024. Exact Bayesian Inference for Loopy Probabilistic Programs using Generating Functions. *Proc. ACM Program. Lang.* 8, OOPSLA1, Article 127 (2024), 31 pages. https://doi.org/10.1145/3649844

Andrey Kofnov, Marcel Moosbrugger, Miroslav Stankovic, Ezio Bartocci, and Efstathia Bura. 2024. Exact and Approximate Moment Derivation for Probabilistic Loops With Non-Polynomial Assignments. *ACM Trans. Model. Comput. Simul.* 34, 3 (2024), 18:1–18:25. https://doi.org/10.1145/3641545

Dexter Kozen. 1981. Semantics of Probabilistic Programs. *J. Comput. Syst. Sci.* 22, 3 (1981), 328–350. https://doi.org/10.1016/0022-0000(81)90036-2

Satoshi Kura, Natsuki Urabe, and Ichiro Hasuo. 2019. Tail Probabilities for Randomized Program Runtimes via Martingales for Higher Moments. In *TACAS 2019: Tools and Algorithms for the Construction and Analysis of Systems - 25th International Conference, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 11428).* Springer, 135–153. https://doi.org/10.1007/978-3-030-17465-1_8

Feynman T. Liang, Liam Hodgkinson, and Michael W. Mahoney. 2023. A Heavy-Tailed Algebra for Probabilistic Programming. In *NeurIPS 2023: Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, New Orleans, LA, USA, December 10-16, 2023.* http://papers.nips.cc/paper_files/paper/2023/hash/3d8f7945cd7f4446cb05a390d4c00558-Abstract-Conference.html

Nicholas D. Matsakis and Felix S. Klock II. 2014. The Rust language. In *HILT 2014: Proceedings of the 2014 ACM SIGAda annual conference on High integrity language technology, Portland, Oregon, USA, October 18-21, 2014.* ACM, 103–104. https://doi.org/10.1145/2663171.2663188

Marcel Moosbrugger, Miroslav Stankovic, Ezio Bartocci, and Laura Kovács. 2022. This is the moment for probabilistic loops. *Proc. ACM Program. Lang.* 6, OOPSLA2 (2022), 1497–1525. https://doi.org/10.1145/3563341

Praveen Narayanan, Jacques Carette, Wren Romano, Chung-chieh Shan, and Robert Zinkov. 2016. Probabilistic Inference by Program Transformation in Hakaru (System Description). In *FLOPS 2016: Functional and Logic Programming - 13th*

International Symposium, Kochi, Japan, March 4-6, 2016, Proceedings (Lecture Notes in Computer Science, Vol. 9613). Springer, 62–79. https://doi.org/10.1007/978-3-319-29604-3_5

Marcel F Neuts. 1975. Probability distributions of phase type. *Liber Amicorum Prof. Emeritus H. Florin* (1975).

Van Chan Ngo, Quentin Carbonneaux, and Jan Hoffmann. 2018. Bounded expectations: resource analysis for probabilistic programs. In *PLDI 2018: Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation, Philadelphia, PA, USA, June 18-22, 2018.* ACM, 496–512. https://doi.org/10.1145/3192366.3192394

Feras A. Saad, Martin C. Rinard, and Vikash K. Mansinghka. 2021. SPPL: probabilistic programming with fast exact symbolic inference. In *PLDI 2021: 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, Virtual Event, Canada, June 20-25, 2021.* ACM, 804–819. https://doi.org/10.1145/3453483.3454078

Jan-Willem van de Meent, Brooks Paige, Hongseok Yang, and Frank Wood. 2018. An Introduction to Probabilistic Programming. *CoRR* abs/1809.10756 (2018). arXiv:1809.10756 http://arxiv.org/abs/1809.10756

Andreas Wächter and Lorenz T. Biegler. 2006. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Math. Program.* 106, 1 (2006), 25–57. https://doi.org/10.1007/S10107-004-0559-Y

Di Wang, Jan Hoffmann, and Thomas W. Reps. 2021. Central moment analysis for cost accumulators in probabilistic programs. In *PLDI 2021: 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, Virtual Event, Canada, June 20-25, 2021.* ACM, 559–573. https://doi.org/10.1145/3453483.3454062

Peixin Wang, Hongfei Fu, Amir Kafshdar Goharshady, Krishnendu Chatterjee, Xudong Qin, and Wenjun Shi. 2019. Cost analysis of nondeterministic probabilistic programs. In *PLDI 2019: Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, Phoenix, AZ, USA, June 22-26, 2019.* ACM, 204–220. https://doi.org/10.1145/3314221.3314581

Peixin Wang, Tengshun Yang, Hongfei Fu, Guanyan Li, and C.-H. Luke Ong. 2024. Static Posterior Inference of Bayesian Probabilistic Programming via Polynomial Solving. *Proc. ACM Program. Lang.* 8, PLDI (2024), 1361–1386. https://doi.org/10.1145/3656432

Fabian Zaiser. 2024a. *Artifact for: Guaranteed Bounds on Posterior Distributions of Discrete Probabilistic Programs with Loops (POPL 2025).* https://doi.org/10.5281/zenodo.14169507

Fabian Zaiser. 2024b. *Towards Formal Verification of Bayesian Inference in Probabilistic Programming via Guaranteed Bounds.* Ph. D. Dissertation. University of Oxford.

Fabian Zaiser, Andrzej S. Murawski, and C.-H. Luke Ong. 2023. Exact Bayesian Inference on Discrete Models via Probability Generating Functions: A Probabilistic Programming Approach. In *NeurIPS 2023: Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, New Orleans, LA, USA, December 10-16, 2023.* http://papers.nips.cc/paper_files/paper/2023/hash/0747af6f877c0cb555fea595f01b0e83-Abstract-Conference.html

Fabian Zaiser, Andrzej S. Murawski, and C.-H. Luke Ong. 2024. Guaranteed Bounds on Posterior Distributions of Discrete Probabilistic Programs with Loops. https://doi.org/10.48550/arXiv.2411.10393 arXiv:2411.10393

Received 2024-07-11; accepted 2024-11-07