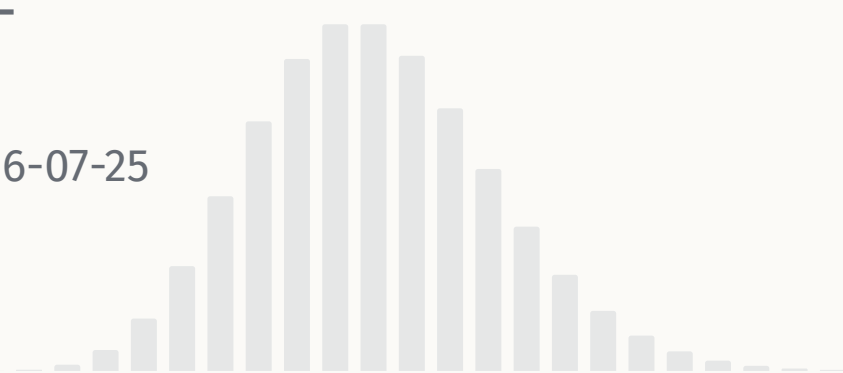


How to Verify Probabilistic Inference — and What That Even Means

Fabian Zaiser · MIT

VERIPROP WORKSHOP · LISBON · 2026-07-25



Example: population survey

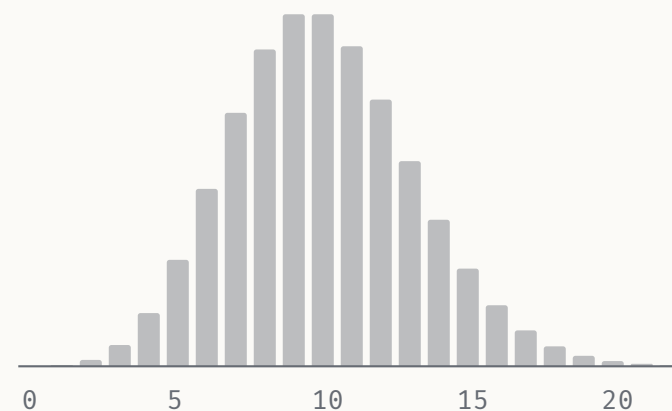
```
N ~ Poisson(20) # unknown
K = 0
for I in 1..N:
    Seen ~ Bernoulli(1/2) # individual I seen w.p. 1/2
    K += Seen
return K
```

randomized program

Example: population survey

```
N ~ Poisson(20) # unknown
K = 0
for I in 1..N:
    Seen ~ Bernoulli(1/2) # individual I seen w.p. 1/2
    K += Seen
return K
```

randomized program



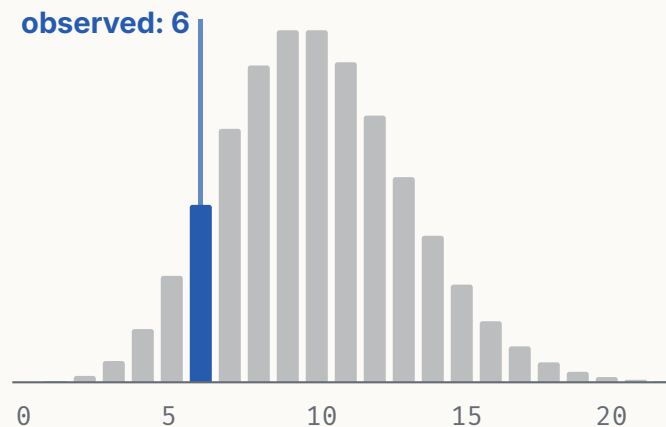
Run **forward**: what's the distribution of count κ ?

Example: population survey

```
N ~ Poisson(20) # unknown
K = 0
for I in 1..N:
    Seen ~ Bernoulli(1/2) # individual I seen w.p. 1/2
    K += Seen

observe K = 6
return N
```

statistical program



Run **forward**: what's the distribution of count K ?

Run **backward**: given $K = 6$, what's the **conditional** distribution of the population N ?

What is probabilistic inference?

Run **forward**: what's the distribution of count κ ?

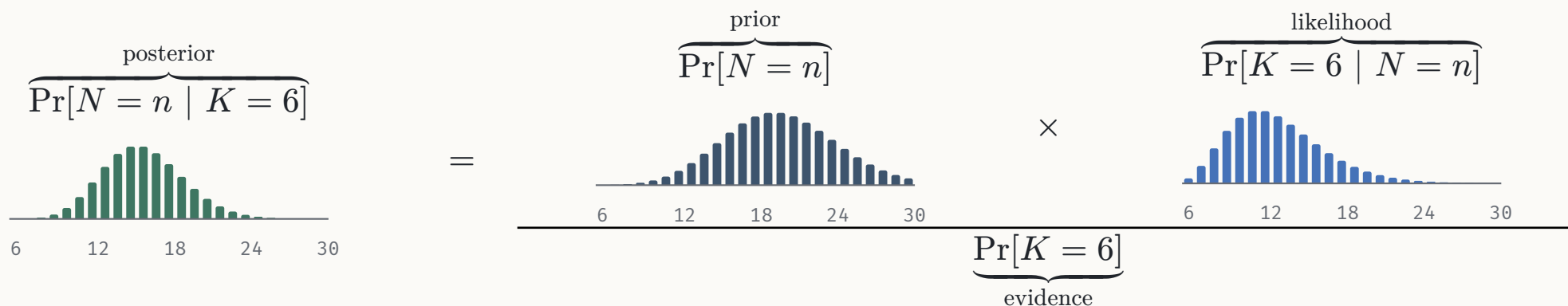
Run **backward**: given $\kappa = 6$, what's the **conditional** distribution of the population N ?

What is probabilistic inference?

Run **forward**: what's the distribution of count κ ?

Run **backward**: given $\kappa = 6$, what's the **conditional** distribution of the population N ?

Bayes' rule:

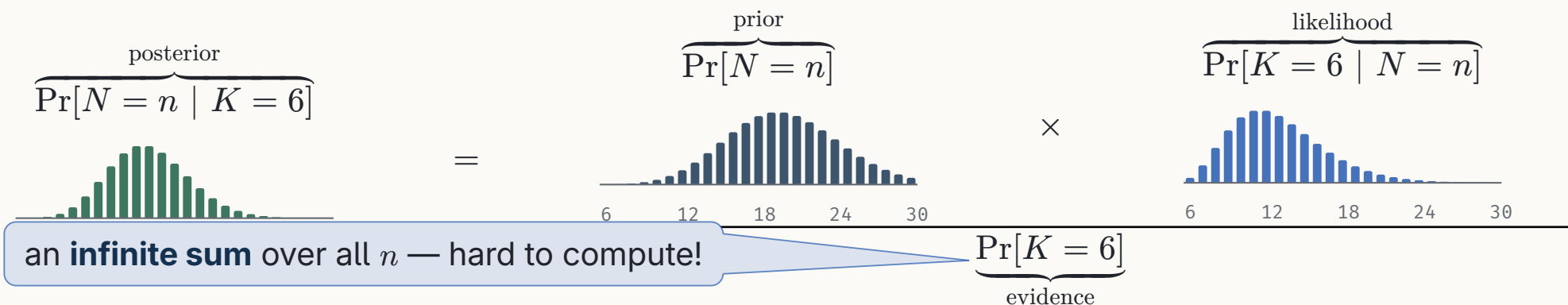


What is probabilistic inference?

Run **forward**: what's the distribution of count κ ?

Run **backward**: given $\kappa = 6$, what's the **conditional** distribution of the population N ?

Bayes' rule:

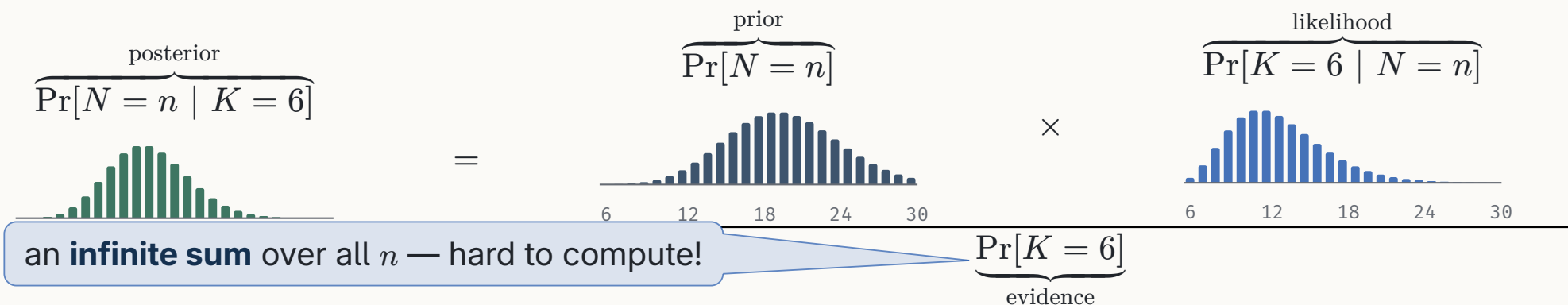


What is probabilistic inference?

Run **forward**: what's the distribution of count κ ?

Run **backward**: given $\kappa = 6$, what's the **conditional** distribution of the population N ?

Bayes' rule:



So how can we compute the posterior?

Exact inference: finite computable representations

→ Sometimes, we can work with a finite representation of probability distributions

Exact inference: finite computable representations

→ Sometimes, we can work with a finite representation of probability distributions

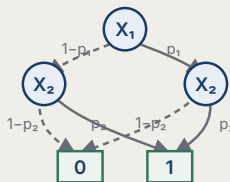
Posterior of the population model: $6 + \text{Poisson}(10)$.

Exact inference: finite computable representations

→ Sometimes, we can work with a finite representation of probability distributions

Posterior of the population model: $6 + \text{Poisson}(10)$.

KNOWLEDGE COMPILATION



DICE, PLUCK

[Holtzen et al., OOPSLA 2020

Bowers et al., PLDI 2025]

✗ only finite support

GENERATING FUNCTIONS

$$G(z) = \sum_n p_n z^n$$

GENFER, PRODIGY

[Zaiser et al., NeurIPS 2023

Klinkenberg et al., OOPSLA 2024]

✗ limited program
operations

SYMBOLIC EXPRESSIONS

$$\int_x f(x) dx$$

HAKARU, PSI, SPPL

[Narayanan et al., FLOPS 2016

Gehr et al., CAV 2016

Saad et al., PLDI 2021]

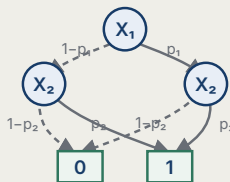
✗ may fail to simplify

Exact inference: finite computable representations

→ Sometimes, we can work with a finite representation of probability distributions

Posterior of the population model: $6 + \text{Poisson}(10)$.

KNOWLEDGE COMPILATION



DICE, PLUCK

[Holtzen et al., OOPSLA 2020

Bowers et al., PLDI 2025]

✗ only finite support

GENERATING FUNCTIONS

$$G(z) = \sum_n p_n z^n$$

GENFER, PRODIGY

[Zaiser et al., NeurIPS 2023

Klinkenberg et al., OOPSLA 2024]

✗ limited program
operations

SYMBOLIC EXPRESSIONS

$$\int_x f(x) dx$$

HAKARU, PSI, SPPL

[Narayanan et al., FLOPS 2016

Gehr et al., CAV 2016

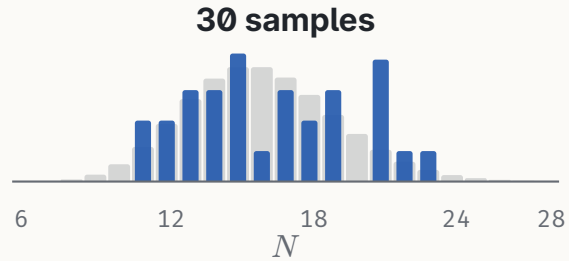
Saad et al., PLDI 2021]

✗ may fail to simplify

Finite exact representations are typically impossible or intractable!

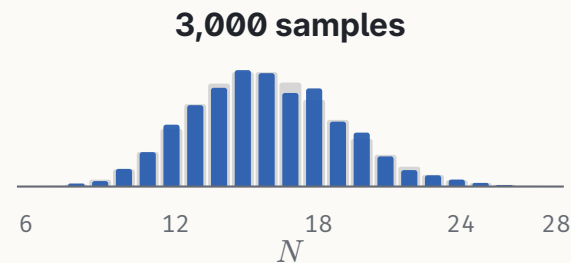
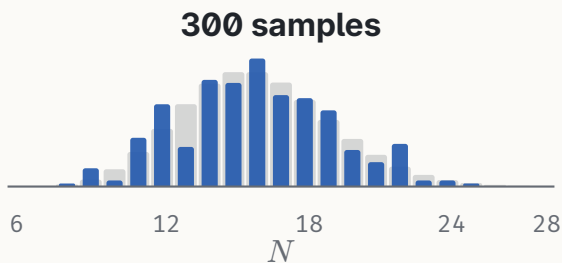
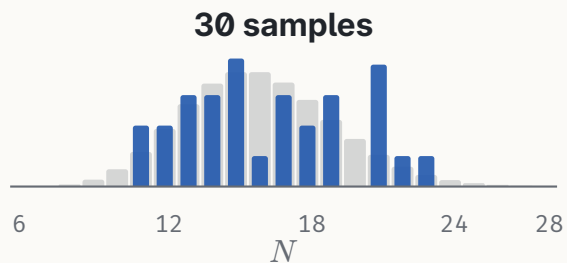
Approximate inference: Monte Carlo algorithms

→ Instead: produce samples whose distribution approximates the posterior



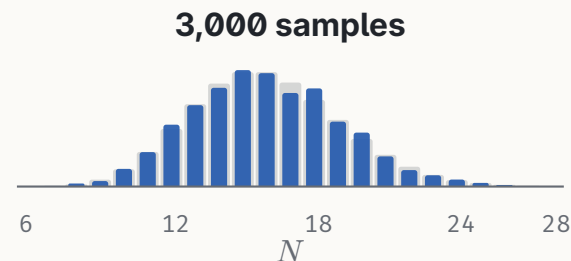
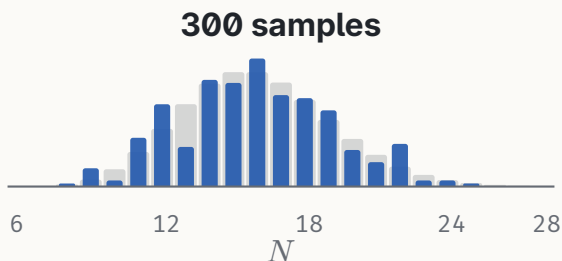
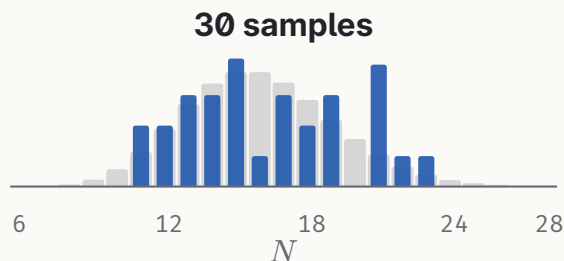
Approximate inference: Monte Carlo algorithms

→ Instead: produce samples whose distribution approximates the posterior



Approximate inference: Monte Carlo algorithms

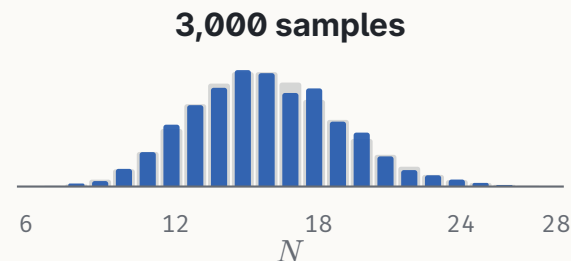
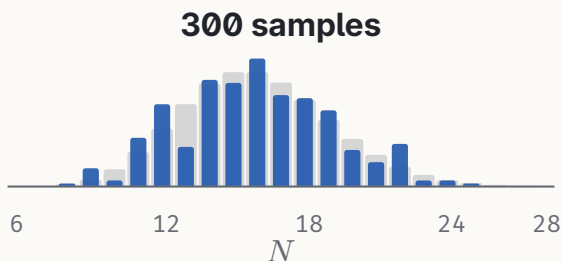
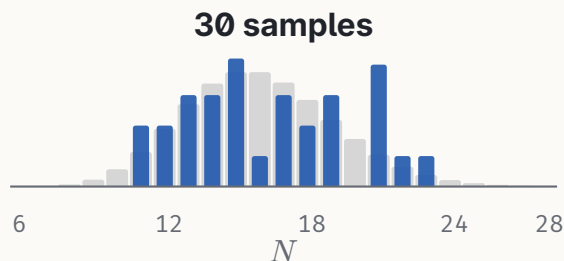
→ Instead: produce samples whose distribution approximates the posterior



Systems:  STAN  PYRO  ANGLICAN PYMC · TURING · GEN · CHURCH · INFER.NET · ...

Approximate inference: Monte Carlo algorithms

→ Instead: produce samples whose distribution approximates the posterior



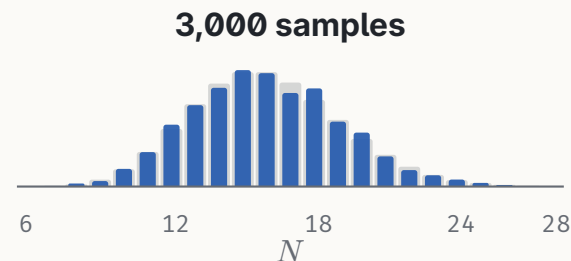
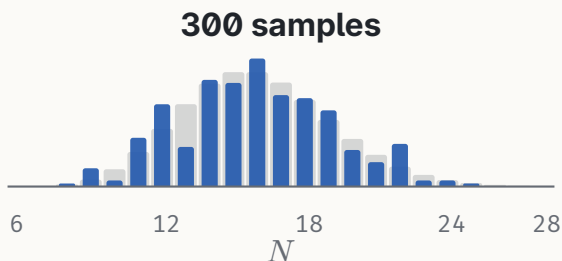
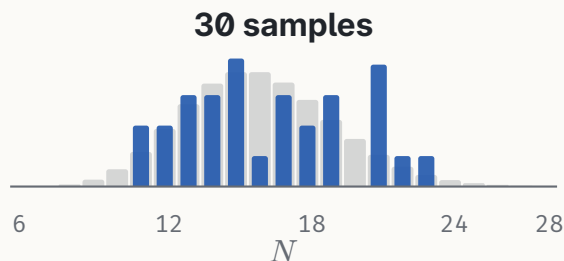
Systems:  STAN  PYRO  ANGLICAN PyMC · TURING · GEN · CHURCH · INFER.NET · ...

How to produce these samples?

- **Markov Chain Monte Carlo (MCMC):** create a Markov chain converging to the posterior (each sample only depends on its predecessor)
- **Sequential Monte Carlo (SMC):** push a population of weighted particles through the program, reweighting and resampling them at observations
- **Variational Inference (VI):** fit the parameters of a simpler distribution to approximate the posterior
- ...

Approximate inference: Monte Carlo algorithms

→ Instead: produce samples whose distribution approximates the posterior



Systems:  STAN  PYRO  ANGLICAN PyMC · TURING · GEN · CHURCH · INFER.NET · ...

How to produce these samples?

focus of this talk

- **Markov Chain Monte Carlo (MCMC):** create a Markov chain converging to the posterior (each sample only depends on its predecessor)
- **Sequential Monte Carlo (SMC):** push a population of weighted particles through the program, reweighting and resampling them at observations
- **Variational Inference (VI):** fit the parameters of a simpler distribution to approximate the posterior
- ...

What does “correct inference” mean?

EXACT INFERENCE

straightforward ✓

APPROXIMATE INFERENCE

???

What does “correct inference” mean?

EXACT INFERENCE

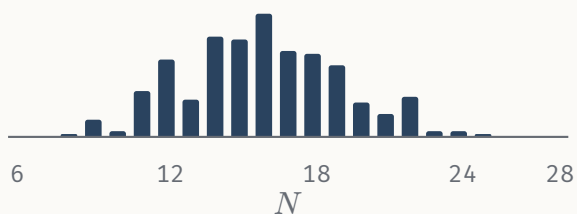
straightforward ✓

APPROXIMATE INFERENCE

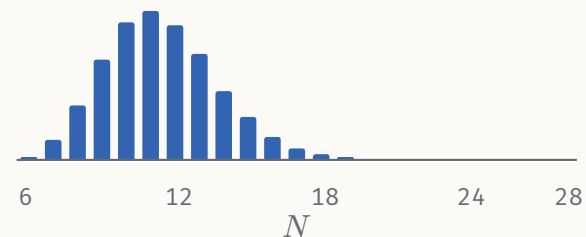
???

[Same program — 300 samples from each of two inference algorithms.]

inference algorithm A



inference algorithm B



What does “correct inference” mean?

EXACT INFERENCE

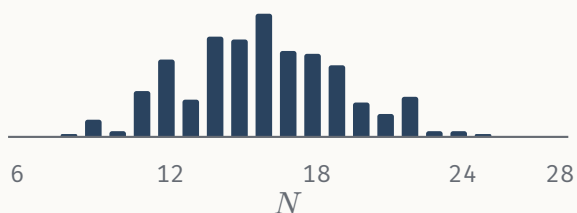
straightforward ✓

APPROXIMATE INFERENCE

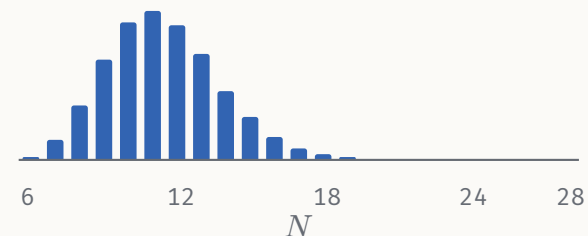
???

[Same program — 300 samples from each of two inference algorithms.]

inference algorithm A



inference algorithm B



- Which (if any) result is correct?

What does “correct inference” mean?

EXACT INFERENCE

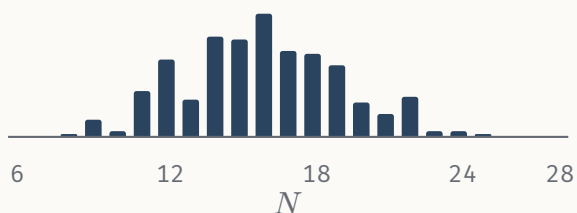
straightforward ✓

APPROXIMATE INFERENCE

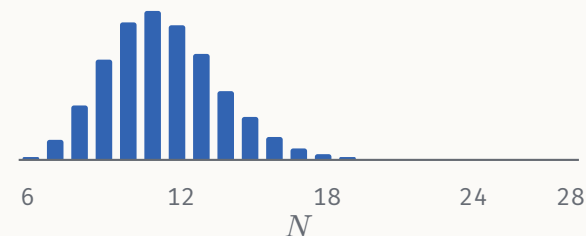
???

[Same program — 300 samples from each of two inference algorithms.]

inference algorithm A



inference algorithm B



- Which (if any) result is correct?
- What does it even mean for an approximation to be correct?

What does “correct inference” mean?

EXACT INFERENCE

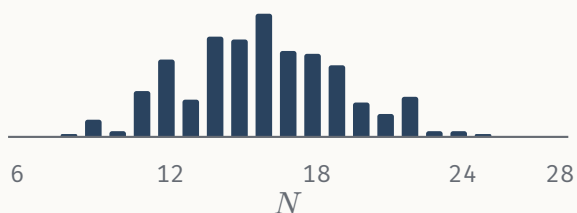
straightforward ✓

APPROXIMATE INFERENCE

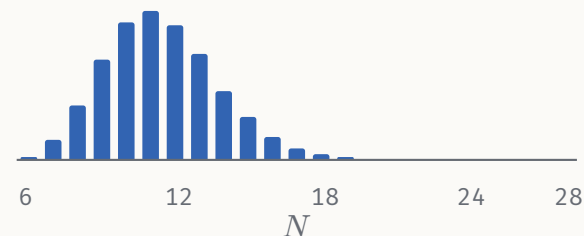
???

[Same program — 300 samples from each of two inference algorithms.]

inference algorithm A



inference algorithm B



- Which (if any) result is correct?
- What does it even mean for an approximation to be correct?
- Is the inference algorithm wrong?

What does “correct inference” mean?

EXACT INFERENCE

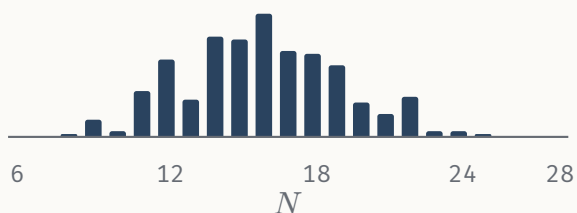
straightforward ✓

APPROXIMATE INFERENCE

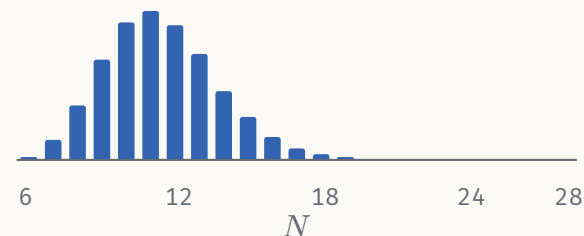
???

[Same program — 300 samples from each of two inference algorithms.]

inference algorithm A



inference algorithm B

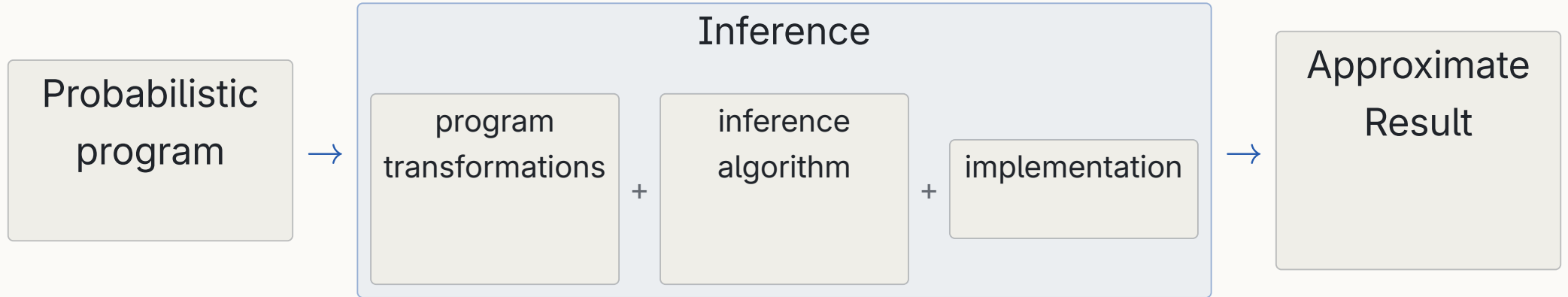


- Which (if any) result is correct?
- What does it even mean for an approximation to be correct?
- Is the inference algorithm wrong?
- Do we just need to run it longer?

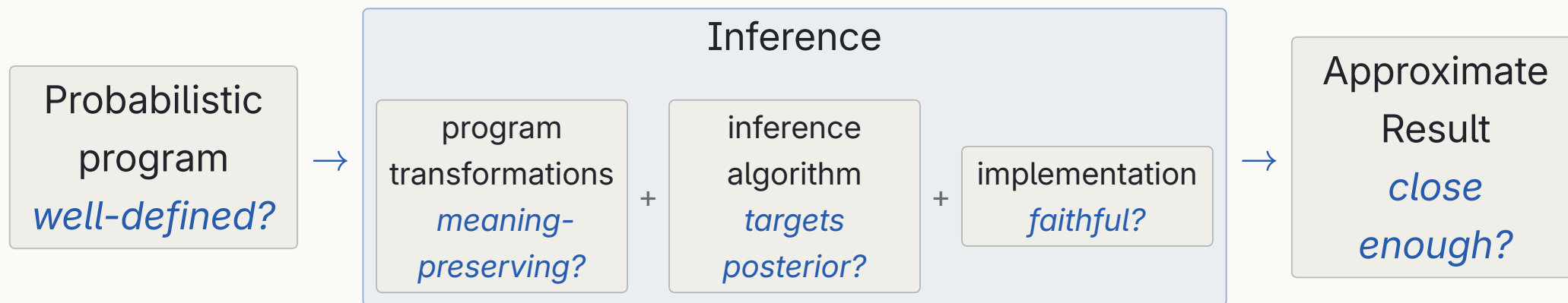
Inference can go wrong in many ways



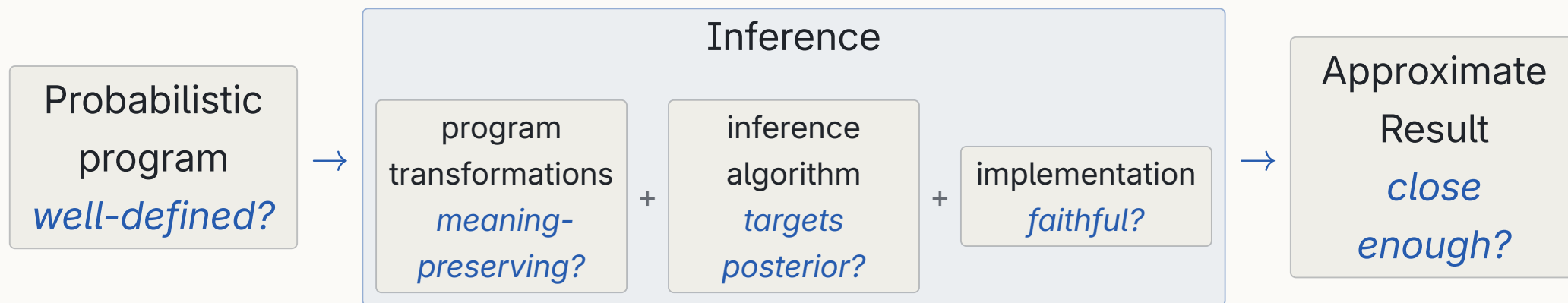
Inference can go wrong in many ways



Inference can go wrong in many ways



Inference can go wrong in many ways



This talk

- overview of techniques to verify all these steps
- deeper dive into some of my work: bounding the distance from the true posterior

Correctness: probabilistic program

probabilistic program

→

transformations

+

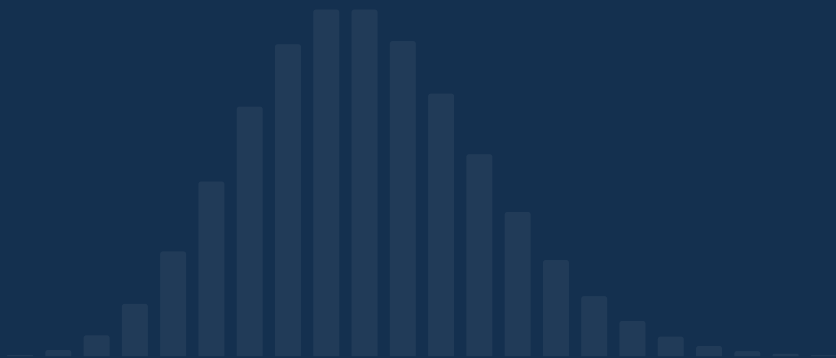
inference algorithm

+

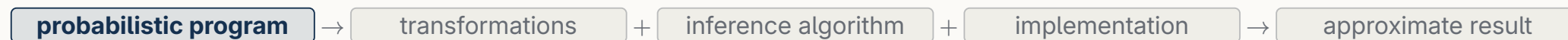
implementation

→

approximate result

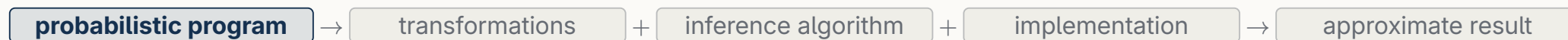


Is my program a good model?



Does the probabilistic program model reality well?

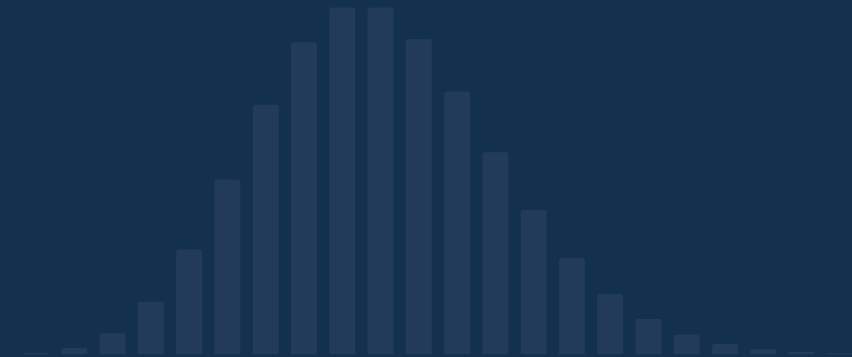
Is my program a good model?



Does the probabilistic program model reality well?

- Statistical checks: prior/posterior predictive checks, sensitivity analysis, ...
- Formal methods for consistency checks: almost sure termination, ...

Correctness: inference algorithm



Does the inference algorithm target the posterior?



Exact inference: prove that the result represents the posterior distribution.

Does the inference algorithm target the posterior?



Exact inference: prove that the result represents the posterior distribution.

Approximate inference: The Markov chain with kernel K targeting posterior π

- **converges** if: $\mu_0 K^n \rightarrow \pi$ as $n \rightarrow \infty$
- is **stationary** on the posterior if: $\pi K = \pi$

Does the inference algorithm target the posterior?



Exact inference: prove that the result represents the posterior distribution.

Approximate inference: The Markov chain with kernel K targeting posterior π

- **converges** if: $\mu_0 K^n \rightarrow \pi$ as $n \rightarrow \infty$
- is **stationary** on the posterior if: $\pi K = \pi$

we want this

Does the inference algorithm target the posterior?



Exact inference: prove that the result represents the posterior distribution.

Approximate inference: The Markov chain with kernel K targeting posterior π

- **converges** if: $\mu_0 K^n \rightarrow \pi$ as $n \rightarrow \infty$
- is **stationary** on the posterior if: $\pi K = \pi$

we want this

that's what is usually proved

Does the inference algorithm target the posterior?



Exact inference: prove that the result represents the posterior distribution.

Approximate inference: The Markov chain with kernel K targeting posterior π

- **converges** if: $\mu_0 K^n \rightarrow \pi$ as $n \rightarrow \infty$
- is **stationary** on the posterior if: $\pi K = \pi$

we want this

that's what is usually proved

EXCEPTION

✓ Borgström et al. [ICFP 2016] prove convergence.

✗ but only for an extremely simple (and impractical) inference algorithm

Preconditions for inference algorithms

Many inference algorithms come with **preconditions**, e.g.:

- **support conditions**: absolute continuity of proposals
→ checked by trace types, guide types

Trace Types and Denotational Semantics for Sound Programmable Inference in Probabilistic Languages

ALEXANDER K. LEW, Massachusetts Institute of Technology, USA
MARCO F. CUSUMANO-TOWNER, Massachusetts Institute of Technology, USA
BENJAMIN SHERMAN, Massachusetts Institute of Technology, USA
MICHAEL CARBIN, Massachusetts Institute of Technology, USA
VIKASH K. MANSINGHKA, Massachusetts Institute of Technology, USA

[Lew et al., POPL 2020]

Sound Probabilistic Inference via Guide Types

Technical Report

Di Wang
Carnegie Mellon University
USA

Jan Hoffmann
Carnegie Mellon University
USA

Thomas Reps
University of Wisconsin
USA

[Wang et al., PLDI 2021]

Preconditions for inference algorithms

Many inference algorithms come with **preconditions**, e.g.:

- **support conditions**: absolute continuity of proposals
→ checked by trace types, guide types
- **differentiability**: some inference algorithms use gradients
→ checked by static analysis

Trace Types and Denotational Semantics for Sound Programmable Inference in Probabilistic Languages

ALEXANDER K. LEW, Massachusetts Institute of Technology, USA
MARCO F. CUSUMANO-TOWNER, Massachusetts Institute of Technology, USA
BENJAMIN SHERMAN, Massachusetts Institute of Technology, USA
MICHAEL CARBIN, Massachusetts Institute of Technology, USA
VIKASH K. MANSINGHA, Massachusetts Institute of Technology, USA

[Lew et al., POPL 2020]

Sound Probabilistic Inference via Guide Types

Technical Report

Di Wang
Carnegie Mellon University
USA

Jan Hoffmann
Carnegie Mellon University
USA

Thomas Reps
University of Wisconsin
USA

[Wang et al., PLDI 2021]

Towards Verified Stochastic Variational Inference for Probabilistic Programs

WONYEOL LEE, School of Computing, KAIST, South Korea
HANGYEOL YU, School of Computing, KAIST, South Korea
XAVIER RIVAL, INRIA Paris and Département d'Informatique of ENS, CNRS/École Normale Supérieure/PSL University, France
HONGSEOK YANG, School of Computing, KAIST, South Korea

[Lee et al., POPL 2020]

Preconditions for inference algorithms

Many inference algorithms come with **preconditions**, e.g.:

- **support conditions**: absolute continuity of proposals
→ checked by trace types, guide types
- **differentiability**: some inference algorithms use gradients
→ checked by static analysis

Trace Types and Denotational Semantics for Sound Programmable Inference in Probabilistic Languages

ALEXANDER K. LEW, Massachusetts Institute of Technology, USA
MARCO F. CUSUMANO-TOWNER, Massachusetts Institute of Technology, USA
BENJAMIN SHERMAN, Massachusetts Institute of Technology, USA
MICHAEL CARBIN, Massachusetts Institute of Technology, USA
VIKASH K. MANSINGHA, Massachusetts Institute of Technology, USA

[Lew et al., POPL 2020]

Sound Probabilistic Inference via Guide Types

Technical Report

Di Wang
Carnegie Mellon University
USA

Jan Hoffmann
Carnegie Mellon University
USA

Thomas Reps
University of Wisconsin
USA

[Wang et al., PLDI 2021]

Towards Verified Stochastic Variational Inference for Probabilistic Programs

WONYEOL LEE, School of Computing, KAIST, South Korea
HANGYEOL YU, School of Computing, KAIST, South Korea
XAVIER RIVAL, INRIA Paris and Département d'Informatique of ENS, CNRS/École Normale Supérieure/PSL University, France
HONGSEOK YANG, School of Computing, KAIST, South Korea

[Lee et al., POPL 2020]

Far from exhaustive!

A widely used inference algorithm was wrong

Lightweight Implementations of Probabilistic Programming Languages Via Transformational Compilation

David Wingate
wingated@mit.edu
Brain and Cognitive Sciences
MIT

Andreas Stuhlmüller
ast@mit.edu
Brain and Cognitive Sciences
MIT

Noah D. Goodman
ngoodman@stanford.edu
Department of Psychology
Stanford

[Wingate, Stuhlmüller & Goodman, AISTATS 2011]

A widely used inference algorithm was wrong

Lightweight Implementations of Probabilistic Programming Languages Via Transformational Compilation

David Wingate
wingated@mit.edu
Brain and Cognitive Sciences
MIT

Andreas Stuhlmüller
ast@mit.edu
Brain and Cognitive Sciences
MIT

Noah D. Goodman
ngoodman@stanford.edu
Department of Psychology
Stanford

[Wingate, Stuhlmüller & Goodman, AISTATS 2011]

INCORRECT (AS PUBLISHED)

domness that was sampled. This gives us an overall score, which is used as an MH accept ratio

$$\alpha = \min \left\{ 1, \frac{p(x') \mathcal{K}_t(x|x', \theta)}{p(x) \mathcal{K}_t(x'|x, \theta)} \right\}$$

CORRECTION (REVISION 3, FEB 2014)

This gives us an overall score, which is used as an MH accept ratio

$$\alpha = \min \left\{ 1, \frac{p(x') \mathcal{K}_t(x|x', \theta) |\mathbb{D}| p_{\text{stale}}}{p(x) \mathcal{K}_t(x'|x, \theta) |\mathbb{D}'| p_{\text{fresh}}} \right\}$$

Correctness: program transformations

probabilistic program

→

transformations

+

inference algorithm

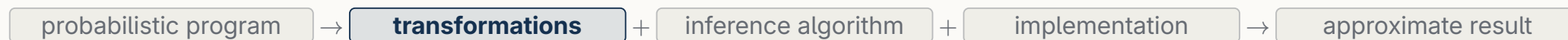
+

implementation

→

approximate result

Program transformations / optimizations



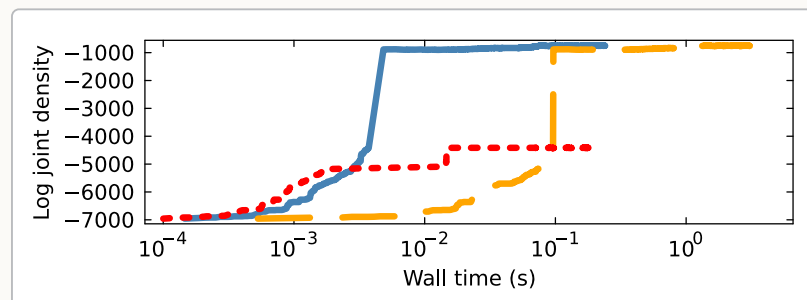
- **Automatic marginalization** [Gorinova et al., TOPLAS 2022]: removes discrete random variables for faster inference
- **Incremental computation** [too many to cite]: makes repeated re-evaluations during inference more efficient

Program transformations / optimizations

probabilistic program → **transformations** + inference algorithm + implementation → approximate result

- **Automatic marginalization** [Gorinova et al., TOPLAS 2022]: removes discrete random variables for faster inference
- **Incremental computation** [too many to cite]: makes repeated re-evaluations during inference more efficient

Incrementalization gone wrong: bug in an existing system

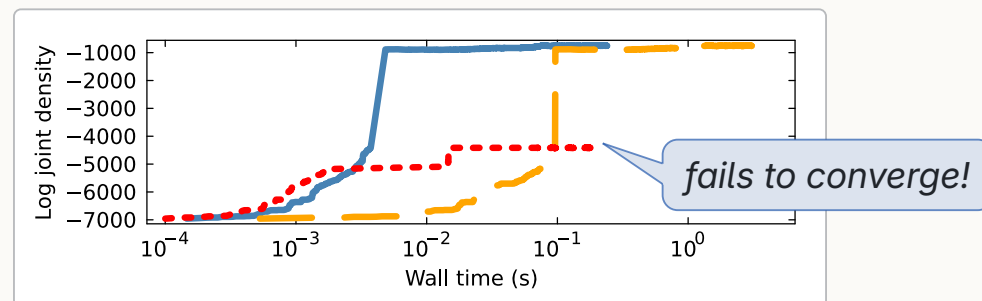


Program transformations / optimizations

probabilistic program → **transformations** + inference algorithm + implementation → approximate result

- **Automatic marginalization** [Gorinova et al., TOPLAS 2022]: removes discrete random variables for faster inference
- **Incremental computation** [too many to cite]: makes repeated re-evaluations during inference more efficient

Incrementalization gone wrong: bug in an existing system



Program transformations / optimizations

probabilistic program → **transformations** + inference algorithm + implementation → approximate result

- **Automatic marginalization** [Gorinova et al., TOPLAS 2022]: removes discrete random variables for faster inference
- **Incremental computation** [too many to cite]: makes repeated re-evaluations during inference more efficient

Incremental Computation for Efficient Programmable Inference in Probabilistic Programs

FABIAN ZAISER, Massachusetts Institute of Technology, USA

JACK CZENSZAK, Yale University, USA

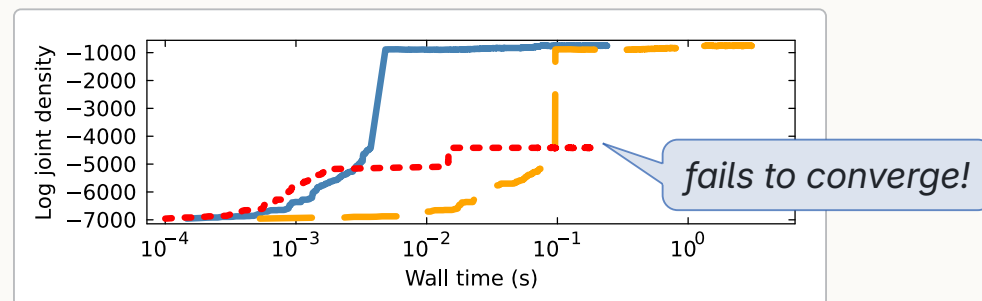
MARTIN C. RINARD, Massachusetts Institute of Technology, USA

VIKASH K. MANSINGHKA, Massachusetts Institute of Technology, USA

ALEXANDER K. LEW, Yale University, USA

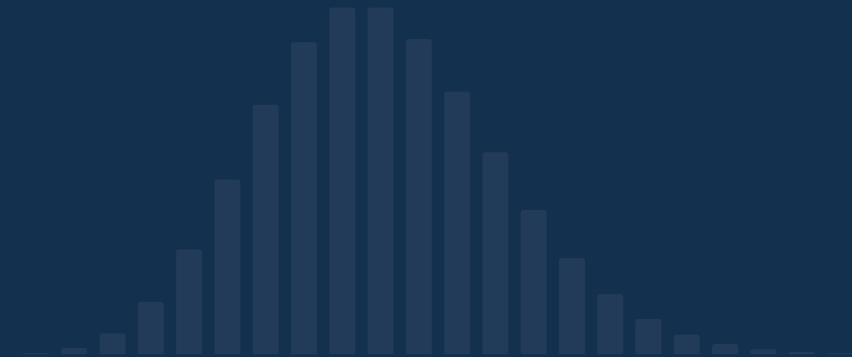
[Zaiser, Czenszak, Rinard, Mansinghka & Lew, PLDI 2026]

Incrementalization gone wrong: bug in an existing system



- reason: ad-hoc incrementalization in many systems
- our solution: **formalize** the program transformation and **prove** it correct!
- allowed for additional optimizations!

Correctness: implementation



probabilistic program



transformations



inference algorithm



implementation



approximate result

Implementation bugs

probabilistic program → transformations + inference algorithm + **implementation** → approximate result

67

new bugs found by
fuzzing EDWARD, PYRO,
and STAN

Testing Probabilistic Programming Systems

Saikat Dutta
University of Illinois, USA
saikatd2@illinois.edu

Owolabi Legunsen
University of Illinois, USA
legunse2@illinois.edu

Zixin Huang
University of Illinois, USA
zixinh2@illinois.edu

Sasa Misailovic
University of Illinois, USA
misailo@illinois.edu

ABSTRACT

Probabilistic programming systems (PP systems) allow developers to model stochastic phenomena and perform efficient inference on the models. The number and adoption of probabilistic programming systems is growing significantly. However, there is no prior study of bugs in these systems and no methodology for systematically testing PP systems. Yet, testing PP systems is highly non-trivial, especially when they perform approximate inference.

In this paper, we characterize 118 previously reported bugs in three open-source PP systems—Edward, Pyro and Stan—and propose *ProbFuzz*, an extensible system for testing PP systems. *ProbFuzz* allows a developer to specify templates of probabilistic models,

others [8, 16, 36, 46, 61, 71, 120]. These systems automate various parts of common inference tasks and support many approximate inference algorithms from machine learning and statistics, including deterministic variational inference and randomized Markov Chain Monte Carlo (MCMC) simulation. Systems like Edward [31, 119] and Pyro [84] embed probabilistic inference within the general deep learning infrastructures, e.g., PyTorch [85] and TensorFlow [117].

A probabilistic programming system (PP system) typically consists of a language, a compiler, and inference procedures. A programmer writes a program in a probabilistic programming language, which extends a standard programming language by adding constructs for (1) random choice, such as sampling from common

[Dutta, Legunsen, Huang & Misailovic, ESEC/FSE 2018]

Implementation bugs



67

new bugs found by
fuzzing EDWARD, PYRO,
and STAN

Most fail loudly.

Testing Probabilistic Programming Systems

Saikat Dutta
University of Illinois, USA
saikatd2@illinois.edu

Owolabi Legunsen
University of Illinois, USA
legunse2@illinois.edu

Zixin Huang
University of Illinois, USA
zixinh2@illinois.edu

Sasa Misailovic
University of Illinois, USA
misailo@illinois.edu

ABSTRACT

Probabilistic programming systems (PP systems) allow developers to model stochastic phenomena and perform efficient inference on the models. The number and adoption of probabilistic programming systems is growing significantly. However, there is no prior study of bugs in these systems and no methodology for systematically testing PP systems. Yet, testing PP systems is highly non-trivial, especially when they perform approximate inference.

In this paper, we characterize 118 previously reported bugs in three open-source PP systems—Edward, Pyro and Stan—and propose *ProbFuzz*, an extensible system for testing PP systems. *ProbFuzz* allows a developer to specify templates of probabilistic models,

others [8, 16, 36, 46, 61, 71, 120]. These systems automate various parts of common inference tasks and support many approximate inference algorithms from machine learning and statistics, including deterministic variational inference and randomized Markov Chain Monte Carlo (MCMC) simulation. Systems like Edward [31, 119] and Pyro [84] embed probabilistic inference within the general deep learning infrastructures, e.g., PyTorch [85] and TensorFlow [117].

A probabilistic programming system (PP system) typically consists of a language, a compiler, and inference procedures. A programmer writes a program in a probabilistic programming language, which extends a standard programming language by adding constructs for (1) random choice, such as sampling from common

[Dutta, Legunsen, Huang & Misailovic, ESEC/FSE 2018]

Implementation bugs



67

new bugs found by
fuzzing EDWARD, PYRO,
and STAN

Most fail loudly.

5

silent accuracy bugs:
*"inaccurate, but
plausible (hard to
catch)"*

Testing Probabilistic Programming Systems

Saikat Dutta
University of Illinois, USA
saikatd2@illinois.edu

Owolabi Legunsen
University of Illinois, USA
legunse2@illinois.edu

Zixin Huang
University of Illinois, USA
zixinh2@illinois.edu

Sasa Misailovic
University of Illinois, USA
misailo@illinois.edu

ABSTRACT

Probabilistic programming systems (PP systems) allow developers to model stochastic phenomena and perform efficient inference on the models. The number and adoption of probabilistic programming systems is growing significantly. However, there is no prior study of bugs in these systems and no methodology for systematically testing PP systems. Yet, testing PP systems is highly non-trivial, especially when they perform approximate inference.

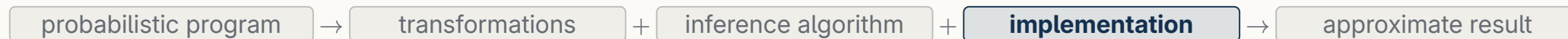
In this paper, we characterize 118 previously reported bugs in three open-source PP systems—Edward, Pyro and Stan—and propose *ProbFuzz*, an extensible system for testing PP systems. ProbFuzz allows a developer to specify templates of probabilistic models,

others [8, 16, 36, 46, 61, 71, 120]. These systems automate various parts of common inference tasks and support many approximate inference algorithms from machine learning and statistics, including deterministic variational inference and randomized Markov Chain Monte Carlo (MCMC) simulation. Systems like Edward [31, 119] and Pyro [84] embed probabilistic inference within the general deep learning infrastructures, e.g., PyTorch [85] and TensorFlow [117].

A probabilistic programming system (PP system) typically consists of a language, a compiler, and inference procedures. A programmer writes a program in a probabilistic programming language, which extends a standard programming language by adding constructs for (1) random choice, such as sampling from common

[Dutta, Legunsen, Huang & Misailovic, ESEC/FSE 2018]

Implementation bugs



67

new bugs found by
fuzzing EDWARD, PYRO,
and STAN

Most fail loudly.

5

silent accuracy bugs:
*"inaccurate, but
plausible (hard to
catch)"*

Testing Probabilistic Programming Systems

Saikat Dutta
University of Illinois, USA
saikatd2@illinois.edu

Owolabi Legunsen
University of Illinois, USA
legunse2@illinois.edu

Zixin Huang
University of Illinois, USA
zixinh2@illinois.edu

Sasa Misailovic
University of Illinois, USA
misailo@illinois.edu

ABSTRACT

Probabilistic programming systems (PP systems) allow developers to model stochastic phenomena and perform efficient inference on the models. The number and adoption of probabilistic programming systems is growing significantly. However, there is no prior study of bugs in these systems and no methodology for systematically testing PP systems. Yet, testing PP systems is highly non-trivial, especially when they perform approximate inference.

In this paper, we characterize 118 previously reported bugs in three open-source PP systems—Edward, Pyro and Stan—and propose *ProbFuzz*, an extensible system for testing PP systems. ProbFuzz allows a developer to specify templates of probabilistic models,

others [8, 16, 36, 46, 61, 71, 120]. These systems automate various parts of common inference tasks and support many approximate inference algorithms from machine learning and statistics, including deterministic variational inference and randomized Markov Chain Monte Carlo (MCMC) simulation. Systems like Edward [31, 119] and Pyro [84] embed probabilistic inference within the general deep learning infrastructures, e.g., PyTorch [85] and TensorFlow [117].

A probabilistic programming system (PP system) typically consists of a language, a compiler, and inference procedures. A programmer writes a program in a probabilistic programming language, which extends a standard programming language by adding constructs for (1) random choice, such as sampling from common

[Dutta, Legunsen, Huang & Misailovic, ESEC/FSE 2018]

Proofs are about $\mathbb{R}$, but implementations use **floating point**.

Verified implementations?

ISABELLE

Source → density

[Eberl, Hölzl & Nipkow, ESOP 2015]

- ✓ verified compiler from programs to density functions
- ✗ sampler using the density is unverified

Verified implementations?

ISABELLE

Source → density

[Eberl, Hölzl & Nipkow, ESOP 2015]

- ✓ verified compiler from programs to density functions
- ✗ sampler using the density is unverified

Rocq

STAN's compiler passes

PROBCOMPCERT

[Tassarotti & Tristan, PLDI 2023]

- ✓ verified density compilation for a real language
- ✗ sampler (NUTS) is unverified

Verified implementations?

ISABELLE

Source → density

[Eberl, Hölzl & Nipkow, ESOP 2015]

- ✓ verified compiler from programs to density functions
- ✗ sampler using the density is unverified

Rocq

STAN's compiler passes

PROBCOMPCERT

[Tassarotti & Tristan, PLDI 2023]

- ✓ verified density compilation for a real language
- ✗ sampler (NUTS) is unverified

Rocq

Source → sample

ZAR

[Bagnall, Stewart & Banerjee, PLDI 2023]

- ✓ verified end-to-end: discrete programs with loops and conditioning
- ✗ no continuous distributions

Verified implementations?

ISABELLE

Source → density

[Eberl, Hölzl & Nipkow, ESOP 2015]

- ✓ verified compiler from programs to density functions
- ✗ sampler using the density is unverified

Rocq

STAN's compiler passes

PROBCOMPCERT

[Tassarotti & Tristan, PLDI 2023]

- ✓ verified density compilation for a real language
- ✗ sampler (NUTS) is unverified

Rocq

Source → sample

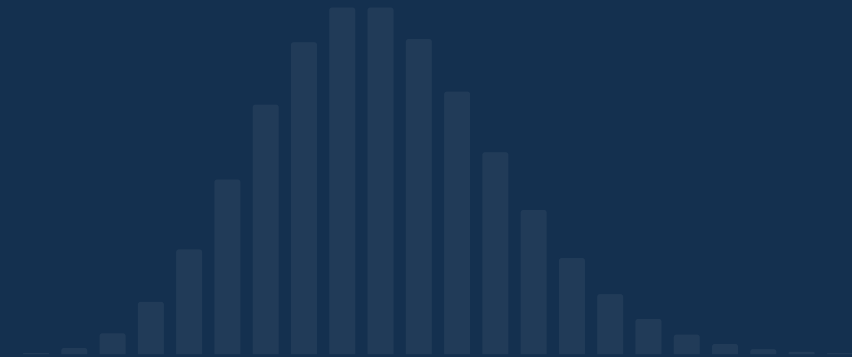
ZAR

[Bagnall, Stewart & Banerjee, PLDI 2023]

- ✓ verified end-to-end: discrete programs with loops and conditioning
- ✗ no continuous distributions

No end-to-end verified system that practitioners actually write their models in.

Correctness: inference result



probabilistic program

→

transformations

+

inference algorithm

+

implementation

→

approximate result

How good is the approximation?

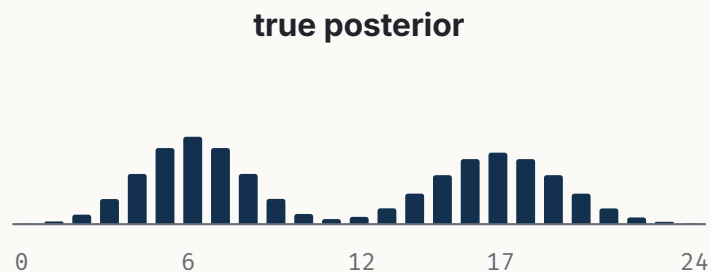


Convergence doesn't tell you how close an approximation is after finite time.

How good is the approximation?



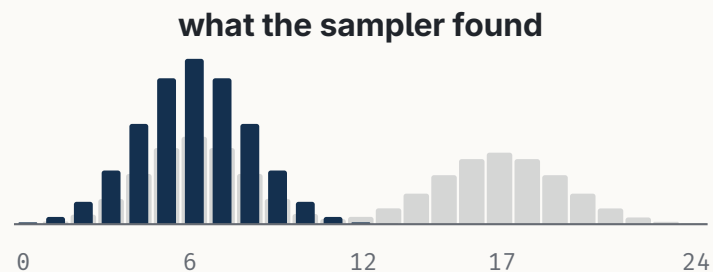
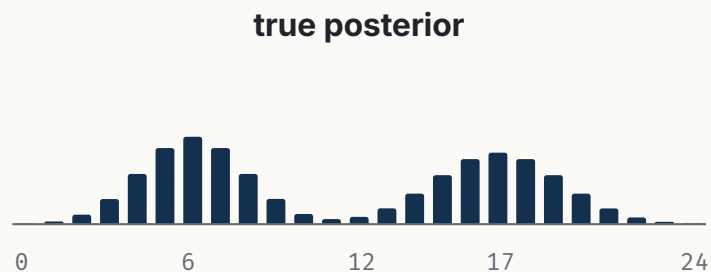
Convergence doesn't tell you how close an approximation is after finite time.



How good is the approximation?



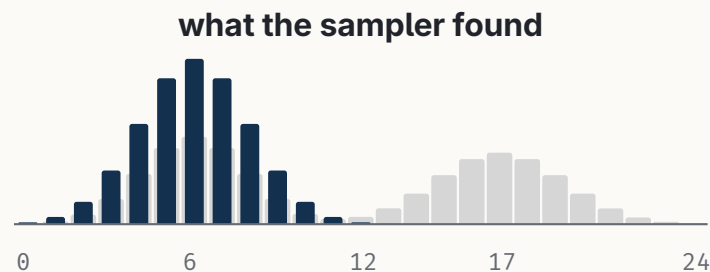
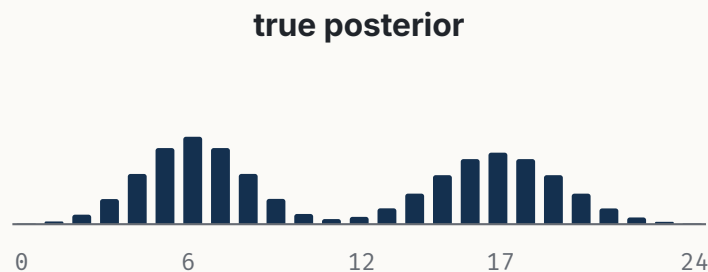
Convergence doesn't tell you how close an approximation is after finite time.



How good is the approximation?



Convergence doesn't tell you how close an approximation is after finite time.



Switching between modes can take exponential time [Mossel & Vigoda, Science 2005]

Statistical diagnostics

Potential scale reduction ($\hat{R}$)

[Gelman & Rubin 1992]

Do independently sampled chains agree?

Effective Sample Size (ESS)

[Vehtari et al. 2021]

How correlated are the samples?

Simulation Based Calibration (SBC)

[Talts et al. 2018]

How well does the sampler perform on simulated data?

Divergences

[Betancourt 2017]

Were there numerical issues during inference?

Statistical diagnostics

Potential scale reduction ($\hat{R}$)

[Gelman & Rubin 1992]

Do independently sampled chains agree?

Effective Sample Size (ESS)

[Vehtari et al. 2021]

How correlated are the samples?

Simulation Based Calibration (SBC)

[Talts et al. 2018]

How well does the sampler perform on simulated data?

Divergences

[Betancourt 2017]

Were there numerical issues during inference?

Diagnostics can tell you something is *wrong* — not that the approximation is *good*.

Statistical diagnostics

Potential scale reduction ($\hat{R}$)

[Gelman & Rubin 1992]

Do independently sampled chains agree?

Effective Sample Size (ESS)

[Vehtari et al. 2021]

How correlated are the samples?

Simulation Based Calibration (SBC)

[Talts et al. 2018]

How well does the sampler perform on simulated data?

Divergences

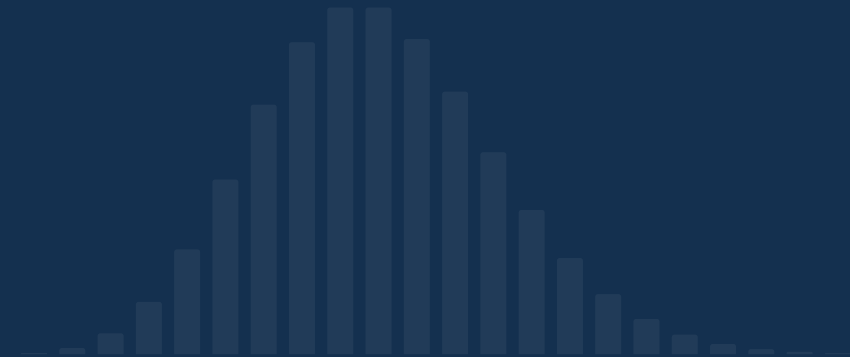
[Betancourt 2017]

Were there numerical issues during inference?

Diagnostics can tell you something is *wrong* — not that the approximation is *good*.

No general-purpose method bounds the approximation error.

Guaranteed bounds



probabilistic program



transformations



inference algorithm



implementation



approximate result

My PhD work: guaranteed bounds

Exact inference

✓ precise result

✗ often intractable

DICE • GENFER • PSI • SPPL

Approximate inference

✓ always applicable

✗ no guarantees

STAN • PYRO • ANGLICAN

My PhD work: guaranteed bounds

Exact inference

- ✓ precise result
- ✗ often intractable

DICE • GENFER • PSI • SPPL

Guaranteed bounds

- ✓ often applicable
- ✓ hard guarantees:

$$\Pr[X = n] \in [l, u]$$

GuBPI • DIABOLO

Approximate inference

- ✓ always applicable
- ✗ no guarantees

STAN • PYRO • ANGLICAN

My PhD work: guaranteed bounds

Exact inference

- ✓ precise result
- ✗ often intractable

DICE • GENFER • PSI • SPPL

Guaranteed bounds

- ✓ often applicable
- ✓ hard guarantees:

$$\Pr[X = n] \in [l, u]$$

GuBPI • DIABOLO

Approximate inference

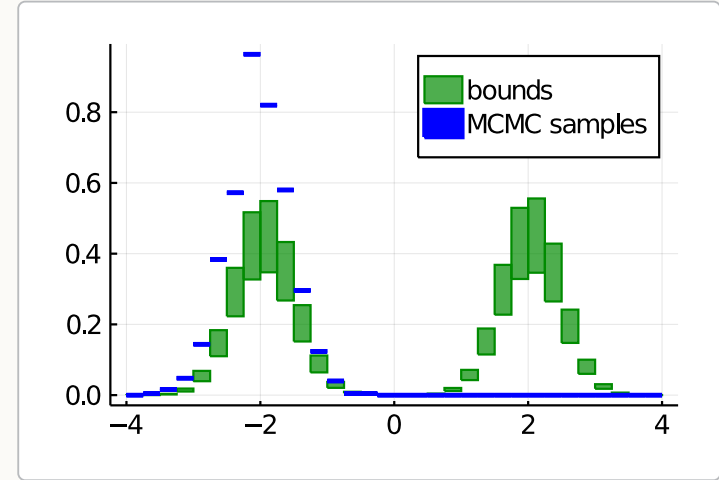
- ✓ always applicable
- ✗ no guarantees

STAN • PYRO • ANGLICAN

There are stochastic certificates (coupling-based bounds [Biswas, Jacob & Vanetti, NeurIPS 2019], bidirectional Monte Carlo [Grosse, Ancha & Roy, NIPS 2016]) but they are themselves estimates, and largely model-specific.

Hard, automatic error bounds for probabilistic programs are rare.

How to find bounds? Use AI (= abstract interpretation)

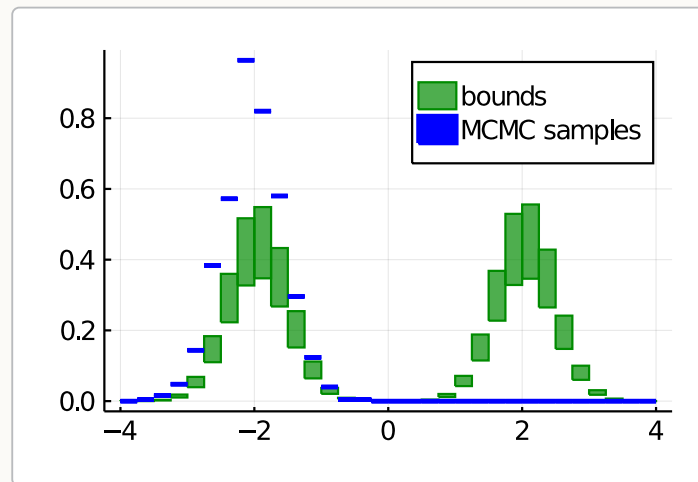


[Beutner, Ong & Zaiser, PLDI 2022]

How to find bounds? Use AI (= abstract interpretation)

Interval domain

- Split the support of distributions into intervals and bound the PDF on each interval.
- Propagate the intervals through the program.



[Beutner, Ong & Zaiser, PLDI 2022]

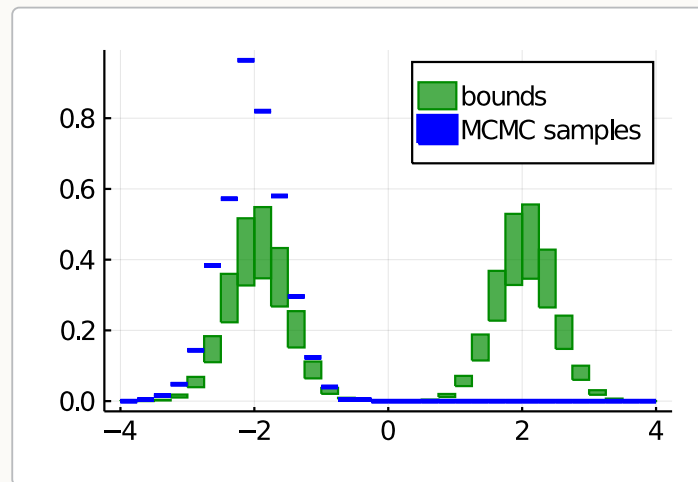
How to find bounds? Use AI (= abstract interpretation)

Interval domain

- Split the support of distributions into intervals and bound the PDF on each interval.
- Propagate the intervals through the program.

Polyhedra

- Finer-grained abstraction for linear guards ...



[Beutner, Ong & Zaiser, PLDI 2022]

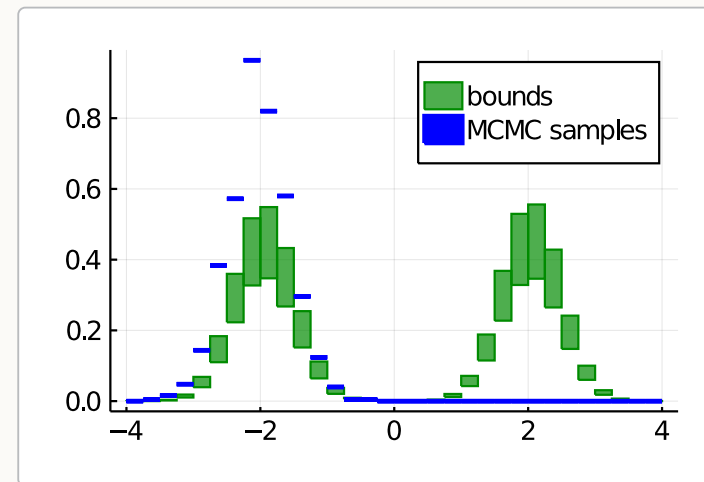
How to find bounds? Use AI (= abstract interpretation)

Interval domain

- Split the support of distributions into intervals and bound the PDF on each interval.
- Propagate the intervals through the program.

Polyhedra

- Finer-grained abstraction for linear guards ...



[Beutner, Ong & Zaiser, PLDI 2022]

WE OBTAIN

- ✓ bounds on posterior probabilities
- ✗ bounds on posterior moments (expectation, variance, ...)
- ✗ tail bounds

DIABOLO: bound the whole distribution

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

DIABOLO: bound the whole distribution

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

RESTRICTION

Only distributions on $\mathbb{N}$, but with loops!

DIABOLO: bound the whole distribution

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

RESTRICTION

Only distributions on $\mathbb{N}$, but with loops!

Bound the whole posterior by

- lower bound: (scaled) finite-support distribution
- upper bound: (scaled) distribution

DIABOLO: bound the whole distribution

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

RESTRICTION

Only distributions on $\mathbb{N}$, but with loops!

Bound the whole posterior by

- lower bound: (scaled) finite-support distribution
- upper bound: (scaled) distribution

What family of (scaled) distributions is suitable for upper bounds?

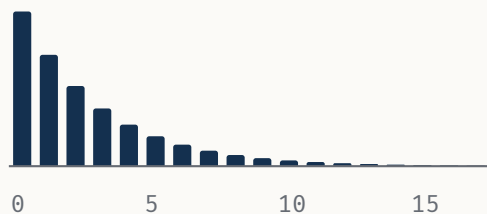
Eventually geometric distributions (EGDs)

→ We need a family of distributions whose **tails** we can compute with

Eventually geometric distributions (EGDs)

→ We need a family of distributions whose **tails** we can compute with

Geometric? $p(n) = \rho(1 - \rho)^n$

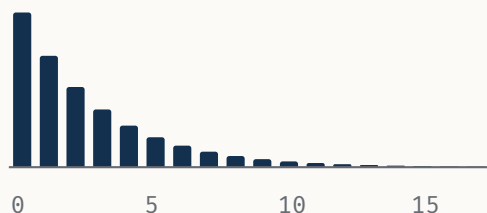


✗ not closed under the program
operations: $x \ += \ 1$ shifts it

Eventually geometric distributions (EGDs)

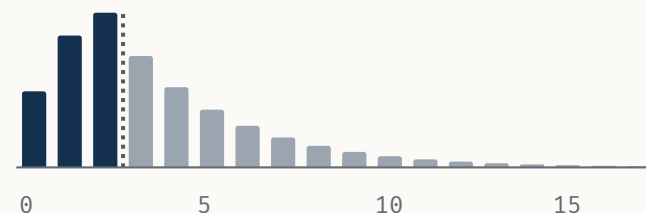
→ We need a family of distributions whose **tails** we can compute with

Geometric? $p(n) = \rho(1 - \rho)^n$



✗ not closed under the program operations: $x \ += \ 1$ shifts it

Eventually geometric $\text{EGD}(Q, \alpha)$: an arbitrary initial block Q , then an exponential tail α^n

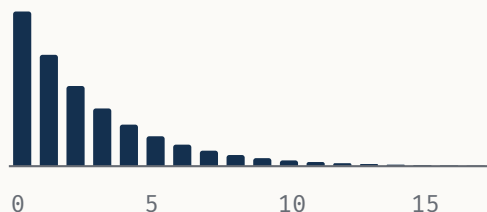


n	0	1	2	3	4	...
$p(n)$	Q_0	Q_1	Q_2	$Q_2 \alpha$	$Q_2 \alpha^2$	...

Eventually geometric distributions (EGDs)

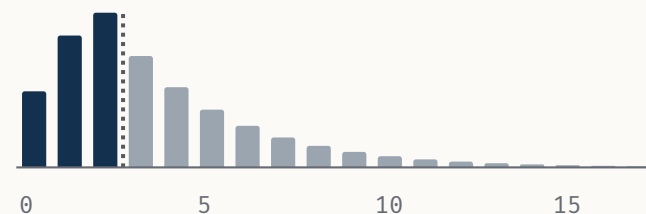
→ We need a family of distributions whose **tails** we can compute with

Geometric? $p(n) = \rho(1 - \rho)^n$



✗ not closed under the program operations: $x += 1$ shifts it

Eventually geometric $\text{EGD}(Q, \alpha)$: an arbitrary initial block Q , then an exponential tail α^n

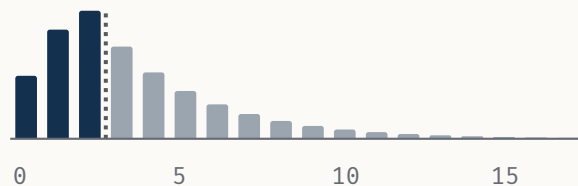


n	0	1	2	3	4	...
$p(n)$	Q_0	Q_1	Q_2	$Q_2\alpha$	$Q_2\alpha^2$	...

- ✓ closed under program operations
- ✓ comparing two EGDs reduces to **polynomial inequalities**

Observations cut the bound down

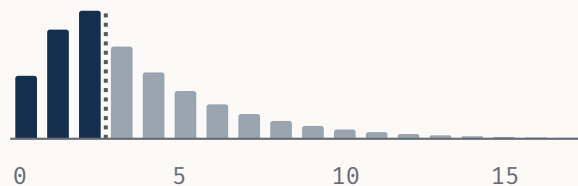
→ Conditioning is the easy part: an observation just zeroes part of the distribution



$$\text{EGD}((Q_0, Q_1, Q_2), \alpha)$$

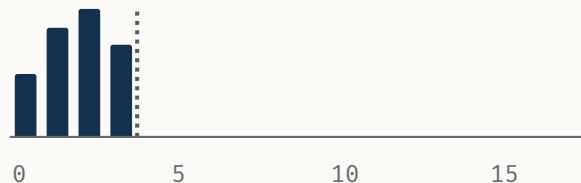
Observations cut the bound down

→ Conditioning is the easy part: an observation just zeroes part of the distribution



$$\text{EGD}((Q_0, Q_1, Q_2), \alpha)$$

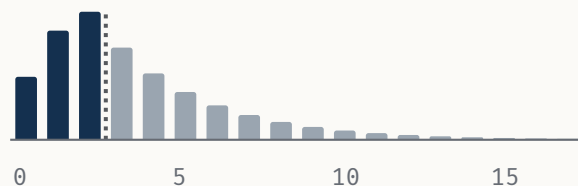
observe $X \leq 3$ ✓



$$\text{EGD}((Q_0, Q_1, Q_2, Q_2\alpha), 0)$$

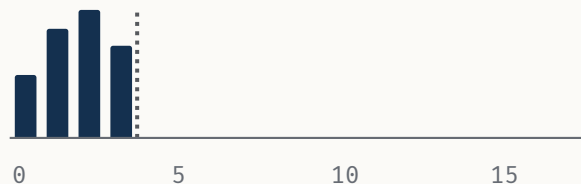
Observations cut the bound down

→ Conditioning is the easy part: an observation just zeroes part of the distribution



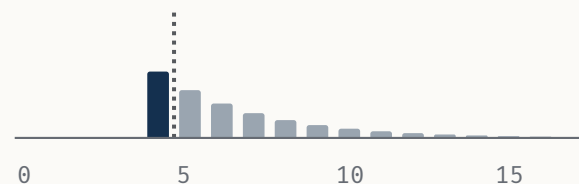
$$EGD((Q_0, Q_1, Q_2), \alpha)$$

observe $X \leq 3 \nwarrow$



$$EGD((Q_0, Q_1, Q_2, Q_2\alpha), 0)$$

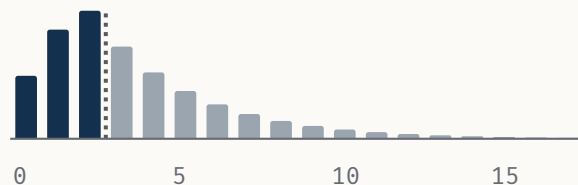
$\searrow$ observe $X \geq 4$



$$EGD((0, 0, 0, 0, Q_2\alpha^2), \alpha)$$

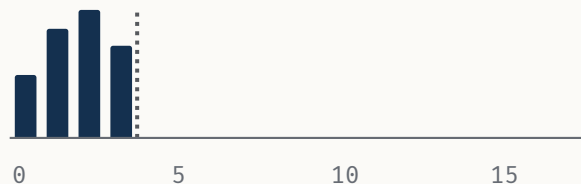
Observations cut the bound down

→ Conditioning is the easy part: an observation just zeroes part of the distribution



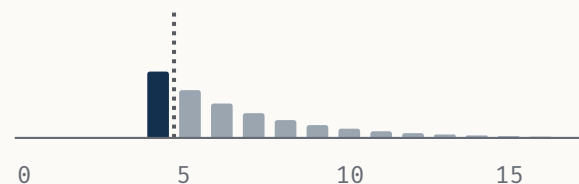
$$EGD((Q_0, Q_1, Q_2), \alpha)$$

observe $X \leq 3$ ↙



$$EGD((Q_0, Q_1, Q_2, Q_2\alpha), 0)$$

↘ observe $X \geq 4$



$$EGD((0, 0, 0, 0, Q_2\alpha^2), \alpha)$$

We work with **unnormalized** measures throughout, and normalize once at the very end — which for bounds means dividing one bound by another.

Loops need an invariant

```
# EGD( $P$ ,  $\alpha$ )  
while cond {  
  # EGD( $P$ ,  $\alpha$ ) | cond  
  body  
  # EGD( $P'$ ,  $\alpha'$ )  
}
```

Loops need an invariant

```
# EGD(P, α)
while cond {
  # EGD(P, α) | cond
  body
  # EGD(P', α')
}
```

If the body contracts by some factor $c < 1$:

$$\text{EGD}(P', \alpha') \leq c \cdot \text{EGD}(P, \alpha)$$

Loops need an invariant

```
# EGD(P, α)
while cond {
  # EGD(P, α) | cond
  body
  # EGD(P', α')
}
# result <= EGD(P/(1-c), α)
```

If the body contracts by some factor $c < 1$:

$$\text{EGD}(P', \alpha') \leq c \cdot \text{EGD}(P, \alpha)$$

then a geometric series bounds **all** iterations at once:

$$\sum_{n=0}^{\infty} c^n \cdot \text{EGD}(P, \alpha) = \frac{1}{1-c} \text{EGD}(P, \alpha)$$

Loops need an invariant

```
# EGD(P, α)
while cond {
  # EGD(P, α) | cond
  body
  # EGD(P', α')
}
# result <= EGD(P/(1-c), α)
```

If the body contracts by some factor $c < 1$:

$$\text{EGD}(P', \alpha') \leq c \cdot \text{EGD}(P, \alpha)$$

then a geometric series bounds **all** iterations at once:

$$\sum_{n=0}^{\infty} c^n \cdot \text{EGD}(P, \alpha) = \frac{1}{1-c} \text{EGD}(P, \alpha)$$

The catch: the bound you arrive with almost never contracts.

Contraction invariants

```
# EGD(P,  $\alpha$ )  
#   <= EGD(Q,  $\beta$ )      the invariant  
while cond {  
  # EGD(Q,  $\beta$ )|cond  
  body  
  # EGD(Q',  $\beta'$ )  
  #   <= c * EGD(Q,  $\beta$ ) contracts!  
}
```

Contraction invariants

```
# EGD(P,  $\alpha$ )
#   <= EGD(Q,  $\beta$ )           the invariant
while cond {
  # EGD(Q,  $\beta$ ) | cond
  body
  # EGD(Q',  $\beta'$ )
  #   <= c * EGD(Q,  $\beta$ )   contracts!
}
```

Find an inductive invariant: don't try to contract the bound you have — first **weaken** it to one that does contract, then apply the geometric series to that one.

Contraction invariants

```
# EGD(P,  $\alpha$ )
#   <= EGD(Q,  $\beta$ )           the invariant
while cond {
  # EGD(Q,  $\beta$ ) | cond
  body
  # EGD(Q',  $\beta'$ )
  #   <= c * EGD(Q,  $\beta$ )   contracts!
}
# result <= EGD(Q/(1-c),  $\beta$ )
```

Find an inductive invariant: don't try to contract the bound you have — first **weaken** it to one that does contract, then apply the geometric series to that one.

More than an ordinary loop invariant: the body must not merely **preserve** the bound, it must **shrink** it by a factor $c < 1$ — and what holds on exit is $1/(1 - c)$ times the invariant, not the invariant itself.

Contraction invariants

```
# EGD(P,  $\alpha$ )
#   <= EGD(Q,  $\beta$ )           the invariant
while cond {
  # EGD(Q,  $\beta$ ) | cond
  body
  # EGD(Q',  $\beta'$ )
  #   <= c * EGD(Q,  $\beta$ )   contracts!
}
# result <= EGD(Q/(1-c),  $\beta$ )
```

Find an inductive invariant: don't try to contract the bound you have — first **weaken** it to one that does contract, then apply the geometric series to that one.

More than an ordinary loop invariant: the body must not merely **preserve** the bound, it must **shrink** it by a factor $c < 1$ — and what holds on exit is $1/(1 - c)$ times the invariant, not the invariant itself.

What is invariant is not a predicate on states, but a bound on a distribution, together with a contraction factor.

How to find the invariant?

1. Run the analysis **symbolically**: leave Q , β and c as unknowns instead of committing to numbers.

How to find the invariant?

1. Run the analysis **symbolically**: leave Q , β and c as unknowns instead of committing to numbers.
2. EGDs are only ever transformed by **rational functions**, so the constraints turn into a system of **polynomial inequalities**.

How to find the invariant?

1. Run the analysis **symbolically**: leave Q , β and c as unknowns instead of committing to numbers.
2. EGDs are only ever transformed by **rational functions**, so the constraints turn into a system of **polynomial inequalities**.
3. Existence of a bound is **decidable**: it is the existential theory of the reals.

How to find the invariant?

1. Run the analysis **symbolically**: leave Q , β and c as unknowns instead of committing to numbers.
2. EGDs are only ever transformed by **rational functions**, so the constraints turn into a system of **polynomial inequalities**.
3. Existence of a bound is **decidable**: it is the existential theory of the reals.

In practice we don't call a decision procedure: a nonlinear solver (IPOPT) proposes a candidate, which we then **verify in exact rational arithmetic**.

What we prove

Soundness. *Every interval we report provably contains the true posterior quantity.*

What we prove

Soundness. *Every interval we report provably contains the true posterior quantity.*

Convergence. *Unroll the loop further and further, and the bounds converge to the exact answer — in total variation, and for every moment.*

What we prove

Soundness. *Every interval we report provably contains the true posterior quantity.*

Convergence. *Unroll the loop further and further, and the bounds converge to the exact answer — in total variation, and for every moment.*

Decidability. *For fixed block sizes, existence of a bound is decidable.*

What we prove

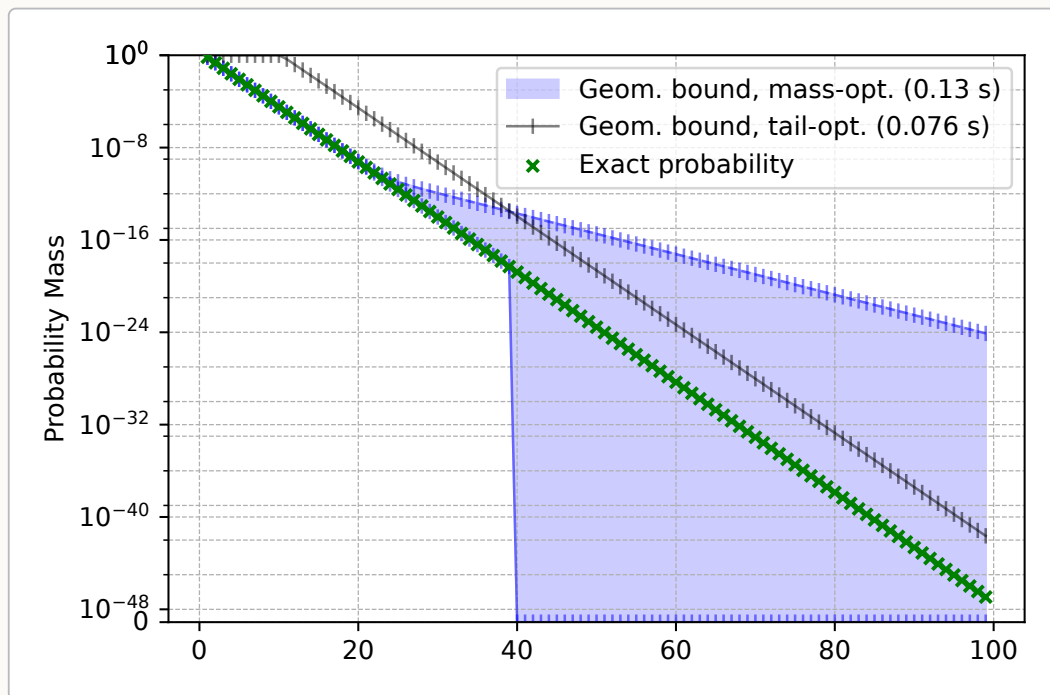
Soundness. *Every interval we report provably contains the true posterior quantity.*

Convergence. *Unroll the loop further and further, and the bounds converge to the exact answer — in total variation, and for every moment.*

Decidability. *For fixed block sizes, existence of a bound is decidable.*

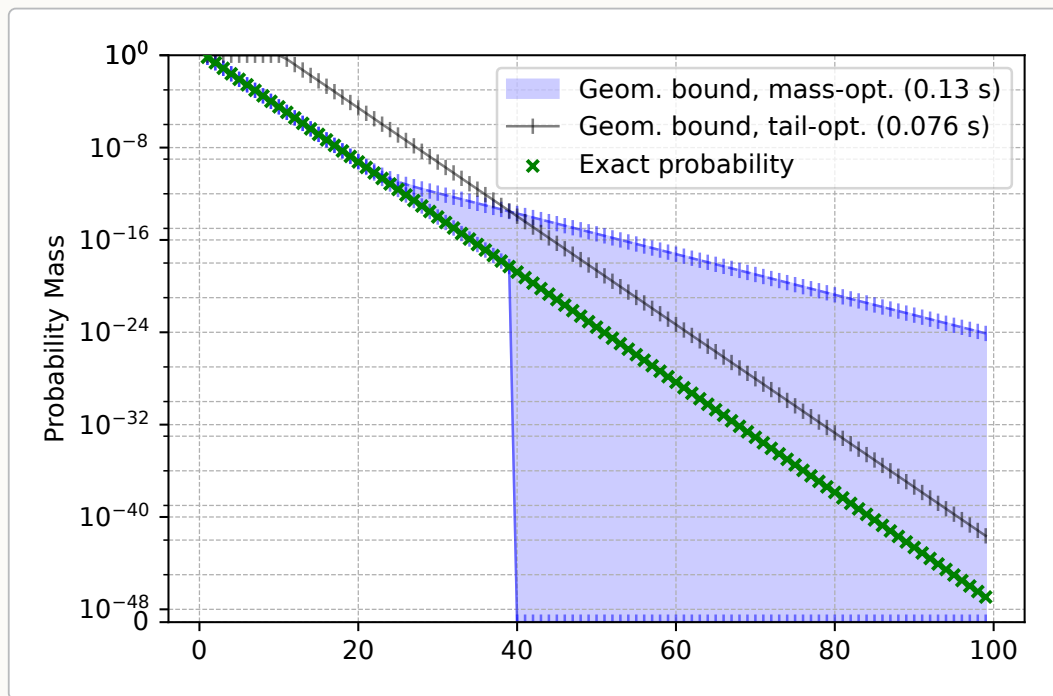
We also proved some necessary and some sufficient conditions for existence of geometric bounds.

And what it delivers



[Zaiser, Murawski & Ong, POPL 2025]

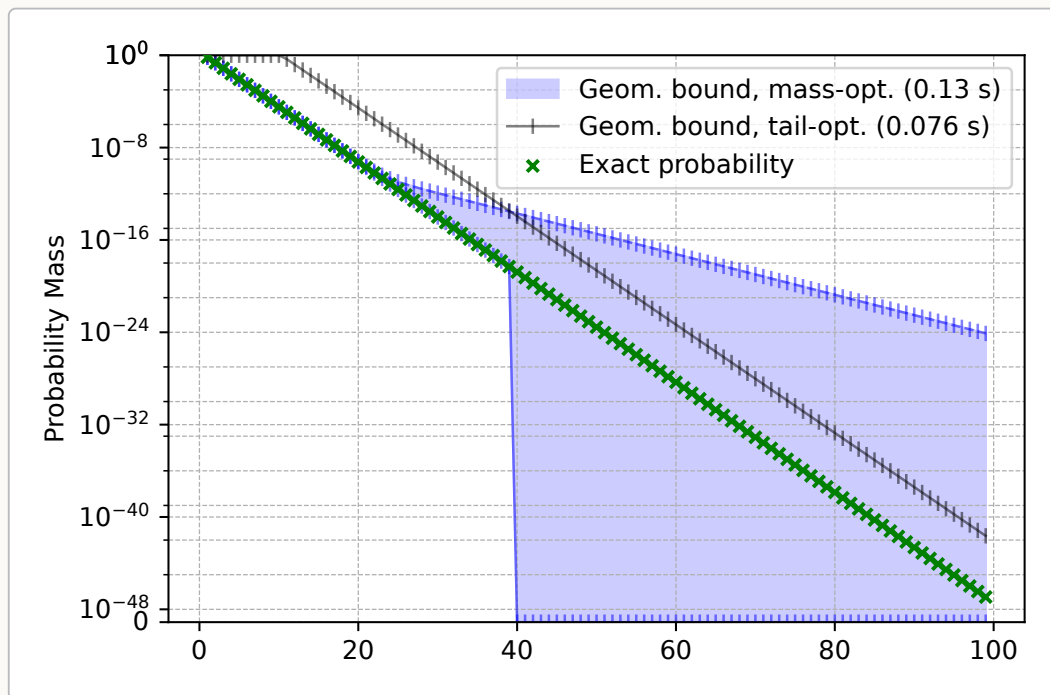
And what it delivers



[Zaiser, Murawski & Ong, POPL 2025]

	expected value	tail
exact	1.5	$\Theta((1/3)^n)$
bounds	$[1.5 - 2 \cdot 10^{-16}, 1.5 + 8 \cdot 10^{-7}]$	$O(0.34^n)$

And what it delivers



[Zaiser, Murawski & Ong, POPL 2025]

	expected value	tail
exact	1.5	$\Theta((1/3)^n)$
bounds	$[1.5 - 2 \cdot 10^{-16}, 1.5 + 8 \cdot 10^{-7}]$	$O(0.34^n)$

Across the benchmark suite:

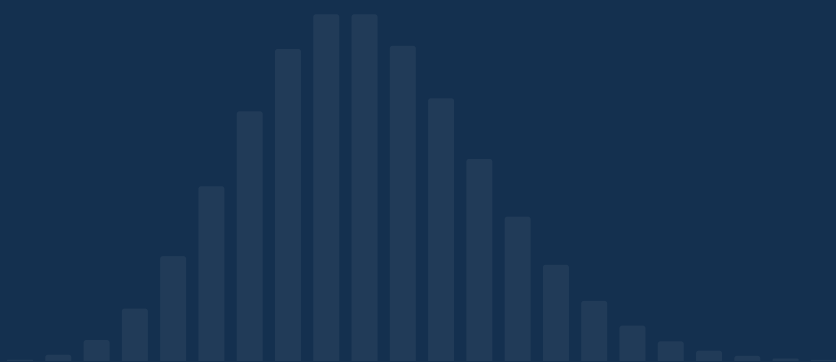
37 / 43

benchmarks
bounded, many
from the literature

≤ 5 min

worst case; usually
seconds

Mechanization



Mechanizing the inference algorithm

WHY VERIFY?

Subtle mathematics

Unstated assumptions

Mechanizing the inference algorithm

WHY VERIFY?

Subtle mathematics

Unstated assumptions

RESPONSE

**Mechanize the inference
algorithm**

Lean · Isabelle · Rocq · ...

Mechanizing the inference algorithm

WHY VERIFY?

Subtle mathematics

Unstated assumptions

RESPONSE

Mechanize the inference algorithm

Lean · Isabelle · Rocq · ...

THEN

Objection: "But this is too much effort!"

Mechanizing the inference algorithm

WHY VERIFY?

Subtle mathematics

Unstated assumptions

RESPONSE

Mechanize the inference algorithm

Lean · Isabelle · Rocq · ...

Now

AI (artificial intelligence, this time)

LLMs make mechanization much cheaper.

Case studies



I had Claude Code formalize the theorems from my last three papers.

Case studies



I had Claude Code formalize the theorems from my last three papers.

I DID ...

- answer the LLM's questions about the paper (clarifications)
- steer the LLM, but at a high-level of abstraction

I DID NOT ...

- write any Lean code
- read the proofs

Case studies



I had Claude Code formalize the theorems from my last three papers.

I DID ...

- answer the LLM's questions about the paper (clarifications)
- steer the LLM, but at a high-level of abstraction

I DID NOT ...

- write any Lean code
- read the proofs

Only the **theorem statements** and the **definitions** they depend on need to be reviewed.

The Lean kernel checks the proofs!

Case studies: results

System / paper	#days	#lines	#lines to review	findings
----------------	-------	--------	---------------------	----------

Case studies: results

System / paper	#days	#lines	#lines to review	findings
GENFER [NeurIPS 2023]	3	4,850	309	pinned down an imprecision in the paper

Case studies: results

System / paper	#days	#lines	#lines to review	findings
GENFER [NeurIPS 2023]	3	4,850	309	pinned down an imprecision in the paper
DIABOLO [POPL 2025]	5	7,923	238	no corrections needed

Case studies: results

System / paper	#days	#lines	#lines to review	findings
GENFER [NeurIPS 2023]	3	4,850	309	pinned down an imprecision in the paper
DIABOLO [POPL 2025]	5	7,923	238	no corrections needed
Incrementalization [PLDI 2026]	17	62,188	7,079	surfaced a few minor errors in the paper

Case studies: results

System / paper	#days	#lines	#lines to review	findings
GENFER [NeurIPS 2023]	3	4,850	309	pinned down an imprecision in the paper
DIABOLO [POPL 2025]	5	7,923	238	no corrections needed
Incrementalization [PLDI 2026]	17	62,188	7,079	surfaced a few minor errors in the paper
GENFER executable (Lean port)	4	+12,225	291	Note: only rational, no floating point approximation

Case studies: results

System / paper	#days	#lines	#lines to review	findings
GENFER [NeurIPS 2023]	3	4,850	309	pinned down an imprecision in the paper
DIABOLO [POPL 2025]	5	7,923	238	no corrections needed
Incrementalization [PLDI 2026]	17	62,188	7,079	surfaced a few minor errors in the paper
GENFER executable (Lean port)	4	+12,225	291	Note: only rational, no floating point approximation

Mechanization has become a lot cheaper. Try it!

Outlook



Open problems: an invitation

INFERENCE ALGORITHM

Convergence

Static analysis to prove convergence on a program

Convergence speed

Program analysis to estimate how quickly it converges

APPROXIMATION RESULT

Guaranteed bounds

Improve scalability; support continuous distributions and observations

IMPLEMENTATION

Floating point

Can we transfer theorems about $\mathbb{R}$ to float?

Testing/fuzzing

Property-based testing for probabilistic systems

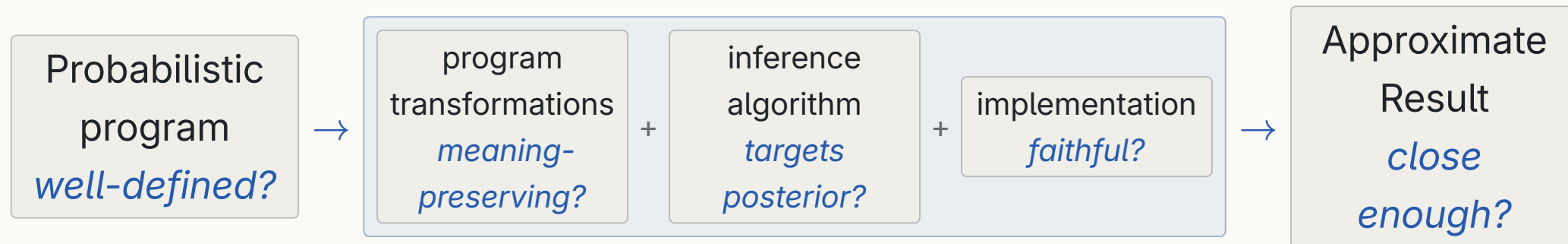
WHOLE PIPELINE

End-to-end mechanization

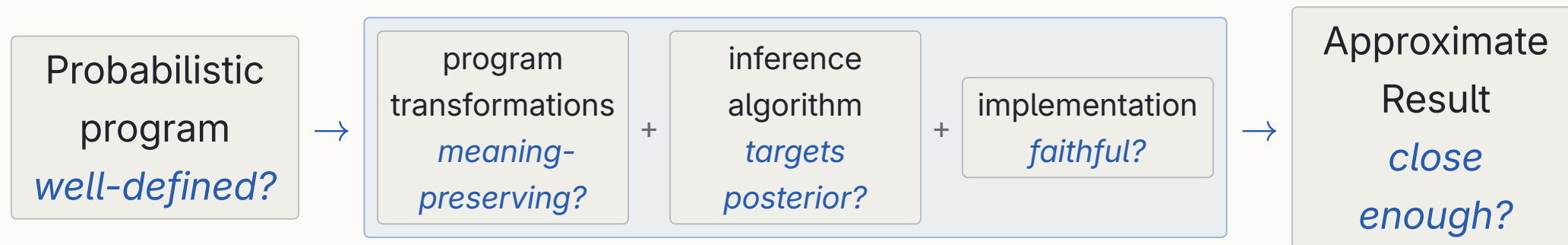
From the probabilistic program to the executable sampler

Lots of interesting problems to solve!

Summary



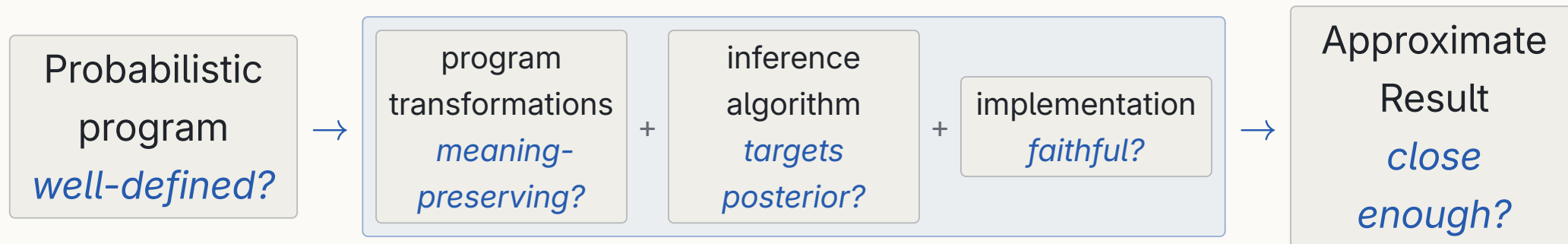
Summary



1

Correctness has many layers — and approximation error remains underexplored.

Summary



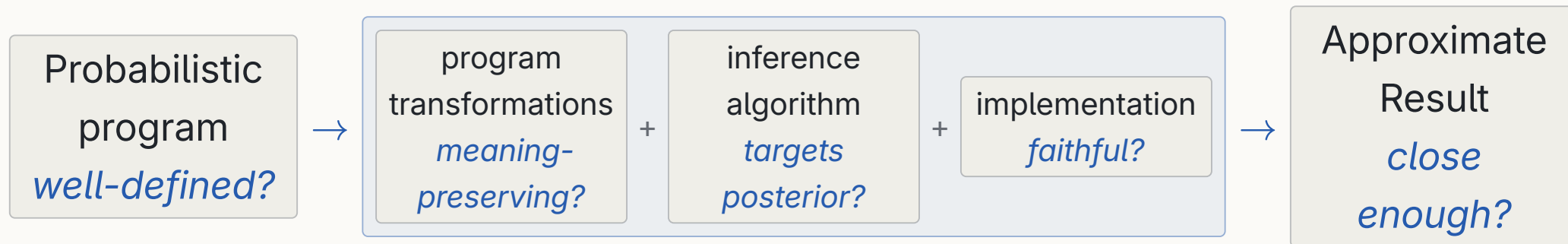
1

Correctness has many layers — and approximation error remains underexplored.

2

Guaranteed bounds
bridge exact and approximate inference.

Summary



1

Correctness has many layers — and approximation error remains underexplored.

2

Guaranteed bounds bridge exact and approximate inference.

3

Formal methods complement statistical checks — and mechanization is now often feasible.