Guaranteed Bounds for Posterior Inference in Universal Probabilistic Programming

Luke Ong

Nanyang Technological University, Singapore   University of Oxford

(Joint work with Raven Beutner and Fabian Zaiser)

What is (Bayesian Statistical) Probabilistic Programming?

Bayes' Rule

$$\mathbb{P}[\theta \mid \mathcal{D}] = \frac{\mathbb{P}[\mathcal{D} \mid \theta] \; \mathbb{P}[\theta]}{\mathbb{P}[\mathcal{D}]}$$

$$\text{Posterior} \;\; \propto \;\; \text{Likelihood} \times \text{Prior}$$

Thomas Bayes (1701-1761)

What is (Bayesian Statistical) Probabilistic Programming?

Bayes' Rule

$$\mathbb{P}[\theta \mid \mathcal{D}] = \frac{\mathbb{P}[\mathcal{D} \mid \theta]\ \mathbb{P}[\theta]}{\mathbb{P}[\mathcal{D}]}$$

Posterior $\propto$ Likelihood $\times$ Prior

Thomas Bayes (1701-1761)

Problem: Probabilistic model development, and the design and implementation of Bayesian (posterior) inference algorithms, are time-consuming and error-prone, often requiring bespoke constructions.

What is (Bayesian Statistical) Probabilistic Programming?

Bayes' Rule

$$\mathbb{P}[\theta \mid \mathcal{D}] = \frac{\mathbb{P}[\mathcal{D} \mid \theta]\ \mathbb{P}[\theta]}{\mathbb{P}[\mathcal{D}]}$$

$$\text{Posterior} \quad \propto \quad \text{Likelihood} \times \text{Prior}$$

Thomas Bayes (1701-1761)

Problem: Probabilistic model development, and the design and implementation of Bayesian (posterior) inference algorithms, are time-consuming and error-prone, often requiring bespoke constructions. This motivates

Probabilistic Programming

– a general-purpose means of expressing probabilistic models as programs, and automatically performing Bayesian inference.

- Probabilistic programming offers an elegant way of generalising graphical models, allowing a much richer representation of models, compositionally.
- Probabilistic programming systems are equipped with implementations of general-purpose inference algorithms.

Vision of Probabilistic Programming

- Expressing probabilistic models as programs: elegant, unifying, potentially benefiting from PL research (semantics and program analysis).
- Availability of general-purpose Bayesian inference engines for arbitrary programs of a universal PPL promotes democratic access to ML algorithms.

Vision of Probabilistic Programming

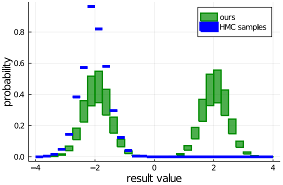
- Expressing probabilistic models as programs: elegant, unifying, potentially benefiting from PL research (semantics and program analysis).
- Availability of general-purpose Bayesian inference engines for arbitrary programs of a universal PPL promotes democratic access to ML algorithms.

What is the Reality?

Unfortunately existing inference algorithms

✗ have few guarantees on the result, and / or

✗ only work on a restricted class of programs (models).



Vision of Probabilistic Programming

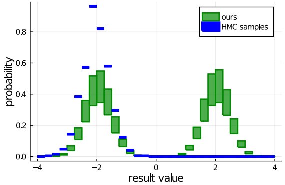
- Expressing probabilistic models as programs: elegant, unifying, potentially benefiting from PL research (semantics and program analysis).
- Availability of general-purpose Bayesian inference engines for arbitrary programs of a universal PPL promotes democratic access to ML algorithms.

What is the Reality?

Unfortunately existing inference algorithms

✗ have few guarantees on the result, and / or

✗ only work on a restricted class of programs (models).



Our Contributions

Guaranteed (nonstochastic and sound) bounds on the posterior distributions.

✓ Diagnostics / (partial) correctness specification: can identify errors in inference results

✓ General applicability: works for a very broad class of probabilistic programs

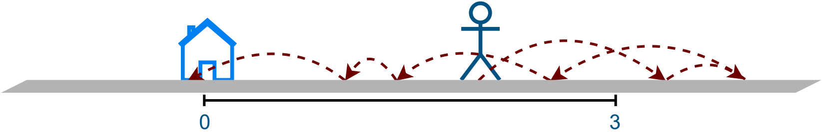
✓ Basis for a new general-purpose inference algorithm (ongoing work).

A Challenging Example Model: Pedestrian

A pedestrian suddenly found himself on a long road. He knows he is no more than 3 km from his home. To get home, he does a random walk: he walks for a random distance no more than 1km, in a random direction (either away or towards his home); and he repeats this `step` until he reaches home. Upon reaching home, his old step counter tells him he has travelled 1.1km.

Question: Where was he (`start`) on the road when he started the random walk?

A Challenging Example Model: Pedestrian

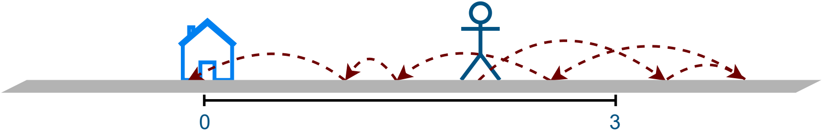


A pedestrian suddenly found himself on a long road. He knows he is no more than 3 km from his home. To get home, he does a random walk: he walks for a random distance no more than 1km, in a random direction (either away or towards his home); and he repeats this `step` until he reaches home. Upon reaching home, his old step counter tells him he has travelled 1.1km.

Question: Where was he (`start`) on the road when he started the random walk?

```
start = sample uniform(0,3)
```

A Challenging Example Model: Pedestrian

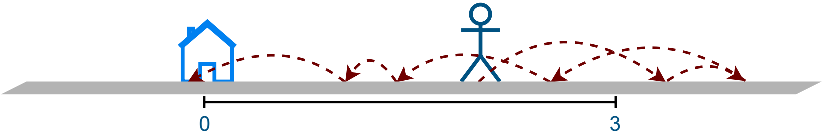


A pedestrian suddenly found himself on a long road. He knows he is no more than 3 km from his home. To get home, he does a random walk: he walks for a random distance no more than 1km, in a random direction (either away or towards his home); and he repeats this `step` until he reaches home. Upon reaching home, his old step counter tells him he has travelled 1.1km.

Question: Where was he (`start`) on the road when he started the random walk?

```
start = sample uniform(0,3)
position = start; distance = 0
while position > 0:
    step = sample uniform(-1, 1)
    position += step
    distance += abs(step)
```

A Challenging Example Model: Pedestrian

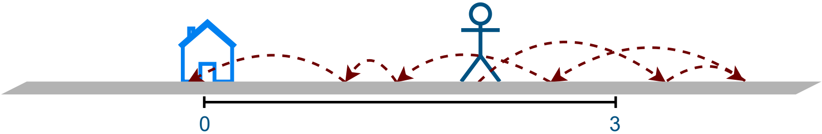


A pedestrian suddenly found himself on a long road. He knows he is no more than 3 km from his home. To get home, he does a random walk: he walks for a random distance no more than 1km, in a random direction (either away or towards his home); and he repeats this `step` until he reaches home. Upon reaching home, his old step counter tells him he has travelled 1.1km.

Question: Where was he (`start`) on the road when he started the random walk?

```
start = sample uniform(0,3)
position = start; distance = 0
while position > 0:
    step = sample uniform(-1, 1)
    position += step
    distance += abs(step)
observe 1.1 from normal(distance, 0.1)
```

A Challenging Example Model: Pedestrian



A pedestrian suddenly found himself on a long road. He knows he is no more than 3 km from his home. To get home, he does a random walk: he walks for a random distance no more than 1km, in a random direction (either away or towards his home); and he repeats this `step` until he reaches home. Upon reaching home, his old step counter tells him he has travelled 1.1km.

Question: Where was he (`start`) on the road when he started the random walk?

```
start = sample uniform(0,3)
position = start; distance = 0
while position > 0:
    step = sample uniform(-1, 1)
    position += step
    distance += abs(step)
observe 1.1 from normal(distance, 0.1)
return start
```

Posterior distribution: $p(start \mid \text{observation})$?

Existing Inference Methods

Existing Inference Methods

1. **Approximate methods:** posterior $\approx X$

- Monte Carlo (particle filter; Metropolis-Hastings, Gibbs sampling, HMC, etc.): "*Through the use of random processes, Metropolis algorithm [Monte Carlo method] offers an efficient way to stumble toward answers to problems that are too complicated to solve exactly.*" (IEEE Computing award citation)
- Optimization-based, notably variational inference

Stan *Pyro* *Anglican*

Existing Inference Methods

1. **Approximate methods:** posterior $\approx X$

- Monte Carlo (particle filter; Metropolis-Hastings, Gibbs sampling, HMC, etc.): "*Through the use of random processes, Metropolis algorithm [Monte Carlo method] offers an efficient way to stumble toward answers to problems that are too complicated to solve exactly.*" (IEEE Computing award citation)
- Optimization-based, notably variational inference

Stan *Pyro* *Anglican*

2. **Exact methods:** posterior $= X$

– Computing a closed-form solution of the posterior inference problem using computer algebra and other forms of symbolic calculations.

PSI SOLVER HAKARU *SPPL*

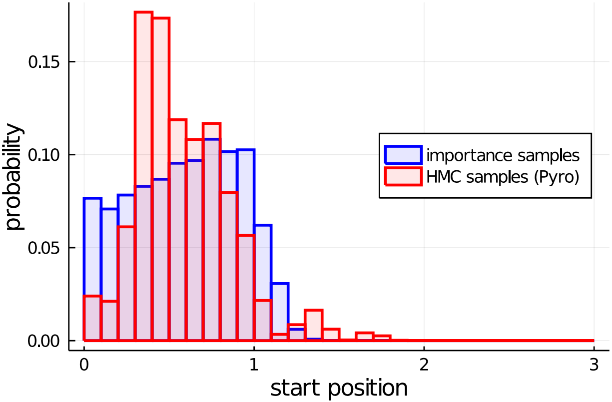
Issues with Existing Methods

- **exact methods:** only work on restricted models (e.g. loop free)
- **approximate methods:** implicit assumptions, slow convergence

Issues with Existing Methods

- **exact methods:** only work on restricted models (e.g. loop free)
- **approximate methods:** implicit assumptions, slow convergence

A (Real) Conundrum: Pedestrian Example



The two distributions are clearly different: at least one is wrong, but which?
(This problem actually sparked and drove the present project.)

Guaranteed Bounds on the Posterior: posterior$_P(E) \in [a, b]$

Desiderata

- A middle ground between exact and approximate methods.
- Given arbitrary program P of a universal PPL with continuous distributions and **`observe`**, and error tolerance, infer guaranteed (nonstochastic and sound) bounds of the posterior: $a \leq \text{posterior}_P(E) \leq b$

Guaranteed Bounds on the Posterior: $\text{posterior}_P(E) \in [a, b]$

Desiderata

- A middle ground between exact and approximate methods.
- Given arbitrary program P of a universal PPL with continuous distributions and `observe`, and error tolerance, infer guaranteed (nonstochastic and sound) bounds of the posterior: $a \leq \text{posterior}_P(E) \leq b$

Why?

- Construct (useful aspects of) ground truth for inference problems.
- Debug (implementations of) approximate inference algorithms.

Guaranteed Bounds on the Posterior: $\text{posterior}_P(E) \in [a, b]$

Desiderata

- A middle ground between exact and approximate methods.
- Given arbitrary program P of a universal PPL with continuous distributions and **`observe`**, and error tolerance, infer guaranteed (nonstochastic and sound) bounds of the posterior: $a \leq \text{posterior}_P(E) \leq b$

Why?

- Construct (useful aspects of) ground truth for inference problems.
- Debug (implementations of) approximate inference algorithms.

How? By exploiting semantics (abstract interpretation) and formal methods

1. interval traces (semantics) & interval arithmetic: basis of the approach
2. constraint-based interval type system: over-approximation of recursive terms
3. stochastic symbolic execution: optimization of special case

Guaranteed Bounds on the Posterior: $\mathsf{posterior}_P(E) \in [a, b]$

Desiderata

- A middle ground between exact and approximate methods.
- Given arbitrary program P of a universal PPL with continuous distributions and **`observe`**, and error tolerance, infer guaranteed (nonstochastic and sound) bounds of the posterior: $a \leq \mathsf{posterior}_P(E) \leq b$

Why?

- Construct (useful aspects of) ground truth for inference problems.
- Debug (implementations of) approximate inference algorithms.

How? By exploiting semantics (abstract interpretation) and formal methods

1. interval traces (semantics) & interval arithmetic: basis of the approach
2. constraint-based interval type system: over-approximation of recursive terms
3. stochastic symbolic execution: optimization of special case

Semantics of Probabilistic Programs

Kozen's Principle

The executions of a probabilistic program are characterised by their traces i.e. the sequence of random draws made during the execution.

Thus a probabilistic program can be interpreted as a deterministic program parametrised by traces.

Semantics of Probabilistic Programs

Kozen's Principle

The executions of a probabilistic program are characterised by their traces i.e. the sequence of random draws made during the execution.

Thus a probabilistic program can be interpreted as a deterministic program parametrised by traces.

- A trace s records **sample**d values, e.g. $\langle 0.23, 0.79, 0.01 \rangle$
- value function: val(s) for trace s
- weight function: weight(s): product of likelihoods of observations

Semantics of Probabilistic Programs

Kozen's Principle

The executions of a probabilistic program are characterised by their traces i.e. the sequence of random draws made during the execution.

Thus a probabilistic program can be interpreted as a deterministic program parametrised by traces.

- A trace s records **sample**d values, e.g. $\langle 0.23, 0.79, 0.01\rangle$
- value function: $\mathsf{val}(s)$ for trace s
- weight function: $\mathsf{weight}(s)$: product of likelihoods of observations

Unnormalized posterior of E (joint distribution of latent and observed variables):

$$[\![P]\!](E) := \int_{\{s \mid \mathsf{val}(s) \in E\}} \mathsf{weight}(s)\,\mathrm{d}s = \text{“}\,\mathbb{P}(\mathtt{start} \in E, \mathsf{obs})\,\text{”}$$

Semantics of Probabilistic Programs

Kozen's Principle

The executions of a probabilistic program are characterised by their traces i.e. the sequence of random draws made during the execution.

Thus a probabilistic program can be interpreted as a deterministic program parametrised by traces.

- A trace $\boldsymbol{s}$ records **sample**d values, e.g. $\langle 0.23, 0.79, 0.01 \rangle$
- value function: $\mathsf{val}(\boldsymbol{s})$ for trace $\boldsymbol{s}$
- weight function: $\mathsf{weight}(\boldsymbol{s})$: product of likelihoods of observations

Unnormalized posterior of E (joint distribution of latent and observed variables):

$$[\![P]\!](E) := \int_{\{\boldsymbol{s} \mid \mathsf{val}(\boldsymbol{s}) \in E\}} \mathsf{weight}(\boldsymbol{s})\, \mathrm{d}\boldsymbol{s} = \text{“}\, \mathbb{P}(\mathtt{start} \in E, \mathsf{obs})\, \text{”}$$

By Bayes' rule, **normalized posterior** (conditional probability):

$$\mathbb{P}(\mathtt{start} \in E \mid \mathsf{obs}) = \frac{\mathbb{P}(\mathtt{start} \in E, \mathsf{obs})}{\mathbb{P}(\mathsf{obs})} = \frac{[\![P]\!](E)}{[\![P]\!](\mathbb{R})}$$

N.B. Normalising constant, $\mathbb{P}(\mathsf{obs})$, is a special case of unnormalised posterior.

Interval Traces

Want to derive bounds of $[\![P]\!](E) := \int_{\{\boldsymbol{s} \mid \mathsf{val}(\boldsymbol{s}) \in E\}} \mathsf{weight}(\boldsymbol{s}) \, \mathrm{d}\boldsymbol{s}$.

Interval Traces

Want to derive bounds of $[\![P]\!](E) := \int_{\{\boldsymbol{s} \mid \mathsf{val}(\boldsymbol{s}) \in E\}} \mathsf{weight}(\boldsymbol{s}) \, \mathrm{d}\boldsymbol{s}$.

Analogy of Riemann integrals: defined by dividing the integral into strips (depending on the error tolerance); and computing the lower sum and upper sum, which sandwich the integral.

Interval Traces

Want to derive bounds of $[\![P]\!](E) := \int_{\{\boldsymbol{s} \mid \mathsf{val}(\boldsymbol{s}) \in E\}} \mathsf{weight}(\boldsymbol{s})\,\mathrm{d}\boldsymbol{s}$.

Analogy of Riemann integrals: defined by dividing the integral into strips (depending on the error tolerance); and computing the lower sum and upper sum, which sandwich the integral.

An interval trace is just a sequence of intervals of the reals; e.g. $\langle [0.1, 0.3], [0.7, 1] \rangle$.

So a set of interval traces summarises a set of traces. E.g. interval trace $\langle [0.1, 0.3], [0.7, 1] \rangle$ contains (or is refined by) traces $\langle 0.2, 0.9 \rangle$ and $\langle 0.23, 0.75 \rangle$.

Idea behind upper bounding $[\![P]\!](E)$:

- Given event E, find a summary (i.e. covering set) $\mathcal{T}$ of interval traces: every $\boldsymbol{s}$ s.t. $\mathsf{val}(\boldsymbol{s}) \in E$ is contained in some interval trace in $\mathcal{T}$.

Interval Traces

Want to derive bounds of $[\![P]\!](E) := \int_{\{\boldsymbol{s} \mid \mathsf{val}(\boldsymbol{s}) \in E\}} \mathsf{weight}(\boldsymbol{s}) \, \mathrm{d}\boldsymbol{s}$.

Analogy of Riemann integrals: defined by dividing the integral into strips (depending on the error tolerance); and computing the lower sum and upper sum, which sandwich the integral.

An interval trace is just a sequence of intervals of the reals; e.g. $\langle [0.1, 0.3], [0.7, 1] \rangle$.

So a set of interval traces summarises a set of traces. E.g. interval trace $\langle [0.1, 0.3], [0.7, 1] \rangle$ contains (or is refined by) traces $\langle 0.2, 0.9 \rangle$ and $\langle 0.23, 0.75 \rangle$.

Idea behind upper bounding $[\![P]\!](E)$:

- Given event E, find a summary (i.e. covering set) $\mathcal{T}$ of interval traces: every $\boldsymbol{s}$ s.t. $\mathsf{val}(\boldsymbol{s}) \in E$ is contained in some interval trace in $\mathcal{T}$.
- Then

$$[\![P]\!](E) \leq \sum_{\boldsymbol{t} \in \mathcal{T}} (\max \mathsf{weight}(\boldsymbol{t})) \, \mathrm{vol}(\boldsymbol{t})$$

Interval Trace Semantics

```
start = sample uniform(0,3)
position = start; distance = 0
while position > 0:
    step = sample uniform(-1, 1)
    position += step
    distance += abs(step)
observe 1.1 from normal(distance, 0.1²)
return start
```

Interval Trace Semantics

```
start = sample uniform(0,3)
position = start; distance = 0
while position > 0:
    step = sample uniform(-1, 1)
    position += step
    distance += abs(step)
observe 1.1 from normal(distance, 0.1²)
return start
```

	standard	interval semantics
`start`		
`position`		
`distance`		
trace $\boldsymbol{s}$	$\langle 0.6\,, 0.2\,, -0.8 \rangle$	$\langle [0.5, 0.6]\,, [0.1, 0.2]\,, [-0.9, -0.8] \rangle$
weight weight$(\boldsymbol{s})$	1	$[1, 1]$
return value val$(\boldsymbol{s})$		

Interval Trace Semantics

```
start = sample uniform(0,3)
position = start; distance = 0
while position > 0:
    step = sample uniform(-1, 1)
    position += step
    distance += abs(step)
observe 1.1 from normal(distance, 0.1²)
return start
```

	standard	interval semantics
`start`	0.6	$[0.5, 0.6]$
`position`		
`distance`		
trace $\boldsymbol{s}$	$\langle 0.6, 0.2, -0.8 \rangle$	$\langle [0.5, 0.6], [0.1, 0.2], [-0.9, -0.8] \rangle$
weight weight$(\boldsymbol{s})$	1	$[1, 1]$
return value val$(\boldsymbol{s})$		

Interval Trace Semantics

```
start = sample uniform(0,3)
position = start; distance = 0
while position > 0:
    step = sample uniform(-1, 1)
    position += step
    distance += abs(step)
observe 1.1 from normal(distance, 0.1²)
return start
```

	standard	interval semantics
start	0.6	$[0.5, 0.6]$
position	0.6	$[0.5, 0.6]$
distance	0.0	$[0.0, 0.0]$
trace $\boldsymbol{s}$	$\langle 0.6\,, 0.2\,, -0.8\rangle$	$\langle [0.5, 0.6]\,, [0.1, 0.2]\,, [-0.9, -0.8]\rangle$
weight weight$(\boldsymbol{s})$	1	$[1, 1]$
return value val$(\boldsymbol{s})$		

Interval Trace Semantics

```
start = sample uniform(0,3)
position = start; distance = 0
while position > 0:
    step = sample uniform(-1, 1)
    position += step
    distance += abs(step)
observe 1.1 from normal(distance, 0.1²)
return start
```

	standard	interval semantics
`start`	0.6	$[0.5, 0.6]$
`position`	0.6	$[0.5, 0.6]$
`distance`	0.0	$[0.0, 0.0]$
trace $\boldsymbol{s}$	$\langle 0.6\,, 0.2\,, -0.8 \rangle$	$\langle [0.5, 0.6]\,, [0.1, 0.2]\,, [-0.9, -0.8] \rangle$
weight weight$(\boldsymbol{s})$	1	$[1, 1]$
return value val$(\boldsymbol{s})$		

Interval Trace Semantics

```
start = sample uniform(0,3)
position = start; distance = 0
while position > 0:
    step = sample uniform(-1, 1)
    position += step
    distance += abs(step)
observe 1.1 from normal(distance, 0.1²)
return start
```

	standard	interval semantics
start	0.6	$[0.5, 0.6]$
position	0.8	$[0.6, 0.8]$
distance	0.2	$[0.1, 0.2]$
trace $\boldsymbol{s}$	$\langle 0.6\,, 0.2\,, -0.8 \rangle$	$\langle [0.5, 0.6]\,, [0.1, 0.2]\,, [-0.9, -0.8] \rangle$
weight weight$(\boldsymbol{s})$	1	$[1, 1]$
return value val$(\boldsymbol{s})$		

Interval Trace Semantics

```
start = sample uniform(0,3)
position = start; distance = 0
while position > 0:
    step = sample uniform(-1, 1)
    position += step
    distance += abs(step)
observe 1.1 from normal(distance, 0.1²)
return start
```

	standard	interval semantics
`start`	0.6	$[0.5, 0.6]$
`position`	0.8	$[0.6, 0.8]$
`distance`	0.2	$[0.1, 0.2]$
trace $\boldsymbol{s}$	$\langle 0.6\,, 0.2\,, -0.8 \rangle$	$\langle [0.5, 0.6]\,, [0.1, 0.2]\,, [-0.9, -0.8] \rangle$
weight weight$(\boldsymbol{s})$	1	$[1, 1]$
return value val$(\boldsymbol{s})$		

Interval Trace Semantics

```
start = sample uniform(0,3)
position = start; distance = 0
while position > 0:
    step = sample uniform(-1, 1)
    position += step
    distance += abs(step)
observe 1.1 from normal(distance, 0.1²)
return start
```

	standard	interval semantics
`start`	0.6	$[0.5, 0.6]$
`position`	0.0	$[-0.3, 0.0]$
`distance`	1.0	$[0.9, 1.1]$
trace $\boldsymbol{s}$	$\langle 0.6\,, 0.2\,, -0.8 \rangle$	$\langle [0.5, 0.6]\,, [0.1, 0.2]\,, [-0.9, -0.8] \rangle$
weight weight$(\boldsymbol{s})$	1	$[1, 1]$
return value val$(\boldsymbol{s})$		

Interval Trace Semantics

```
start = sample uniform(0,3)
position = start; distance = 0
while position > 0:
    step = sample uniform(-1, 1)
    position += step
    distance += abs(step)
observe 1.1 from normal(distance, 0.1²)
return start
```

	standard	interval semantics
`start`	0.6	$[0.5, 0.6]$
`position`	0.0	$[-0.3, 0.0]$
`distance`	1.0	$[0.9, 1.1]$
trace $\boldsymbol{s}$	$\langle 0.6\,, 0.2\,, -0.8 \rangle$	$\langle\, [0.5, 0.6]\,, [0.1, 0.2]\,, [-0.9, -0.8] \,\rangle$
weight weight$(\boldsymbol{s})$	≈ 2.4	$[0.53, 3.99]$
return value val$(\boldsymbol{s})$		

Interval Trace Semantics

```
start = sample uniform(0,3)
position = start; distance = 0
while position > 0:
    step = sample uniform(-1, 1)
    position += step
    distance += abs(step)
observe 1.1 from normal(distance, 0.1²)
return start
```

	standard	interval semantics
`start`	0.6	$[0.5, 0.6]$
`position`	0.0	$[-0.3, 0.0]$
`distance`	1.0	$[0.9, 1.1]$
trace $\boldsymbol{s}$	$\langle 0.6\,, 0.2\,, -0.8 \rangle$	$\langle [0.5, 0.6]\,, [0.1, 0.2]\,, [-0.9, -0.8] \rangle$
weight weight$(\boldsymbol{s})$	≈ 2.4	$[0.53, 3.99]$
return value val$(\boldsymbol{s})$	0.6	$[0.5, 0.6]$

Robustness, Precision and Effectivity

Soundness

For all non-overlapping and exhaustive set of interval traces $\mathcal{T}$:

$$\mathsf{lowerBd}_P^{\mathcal{T}} \leq [\![P]\!] \leq \mathsf{upperBd}_P^{\mathcal{T}}.$$

$\mathsf{lowerBd}_P^{\mathcal{T}}$ and $\mathsf{upperBd}_P^{\mathcal{T}}$ are super-/sub-additive measures.

Robustness, Precision and Effectivity

Soundness

For all non-overlapping and exhaustive set of interval traces $\mathcal{T}$:

$$\mathsf{lowerBd}_P^{\mathcal{T}} \leq [\![P]\!] \leq \mathsf{upperBd}_P^{\mathcal{T}}.$$

$\mathsf{lowerBd}_P^{\mathcal{T}}$ and $\mathsf{upperBd}_P^{\mathcal{T}}$ are super-/sub-additive measures.

Completeness

For all intervals I and $\epsilon > 0$, there is a countable set $\mathcal{T}$ of (non-overlapping and exhaustive) interval traces s.t.

$$\mathsf{upperBd}_P^{\mathcal{T}}(I) - \epsilon \leq [\![P]\!](I) \leq \mathsf{lowerBd}_P^{\mathcal{T}}(I) + \epsilon$$

under the assumptions:

- the primitive functions are continuous*
- each **`sample`**d value is used at most once in each conditional, **`observe`** statement, and in the return value.

Robustness, Precision and Effectivity

Soundness

For all non-overlapping and exhaustive set of interval traces $\mathcal{T}$:

$$\mathsf{lowerBd}_P^{\mathcal{T}} \leq [\![P]\!] \leq \mathsf{upperBd}_P^{\mathcal{T}}.$$

$\mathsf{lowerBd}_P^{\mathcal{T}}$ and $\mathsf{upperBd}_P^{\mathcal{T}}$ are super-/sub-additive measures.

Completeness

For all intervals I and $\epsilon > 0$, there is a countable set $\mathcal{T}$ of (non-overlapping and exhaustive) interval traces s.t.

$$\mathsf{upperBd}_P^{\mathcal{T}}(I) - \epsilon \leq [\![P]\!](I) \leq \mathsf{lowerBd}_P^{\mathcal{T}}(I) + \epsilon$$

under the assumptions:

- the primitive functions are continuous*
- each **`sample`**d value is used at most once in each conditional, **`observe`** statement, and in the return value.

Robustness, Precision and Effectivity

Soundness

For all non-overlapping and exhaustive set of interval traces $\mathcal{T}$:

$$\mathsf{lowerBd}_P^{\mathcal{T}} \leq [\![P]\!] \leq \mathsf{upperBd}_P^{\mathcal{T}}.$$

$\mathsf{lowerBd}_P^{\mathcal{T}}$ and $\mathsf{upperBd}_P^{\mathcal{T}}$ are super-/sub-additive measures.

Completeness

For all intervals I and $\epsilon > 0$, there is a finite set $\mathcal{T}$ of (non-overlapping and exhaustive) interval traces s.t.

$$[\![P]\!](I) \leq \mathsf{lowerBd}_P^{\mathcal{T}}(I) + \epsilon$$

under the assumptions:

- the primitive functions are continuous*
- each **`sample`**d value is used at most once in each conditional, **`observe`** statement, and in the return value.

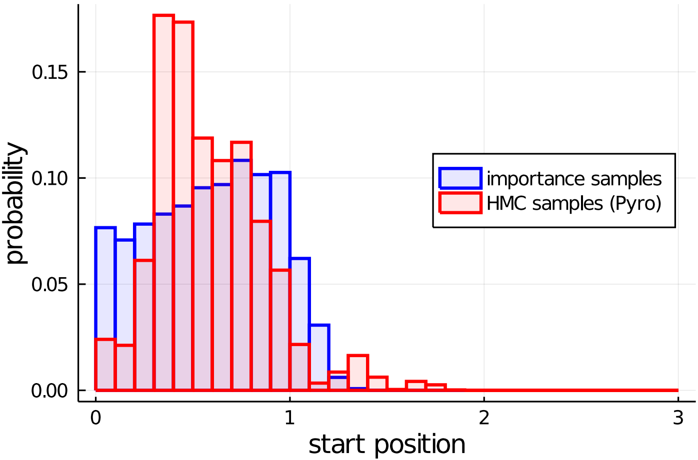
Empirical Evaluation

- Implementation: GuBPI (Guaranteed Bounds for Posterior inference)
  gubpi-tool.github.io

Empirical Evaluation

- Implementation: GuBPI (Guaranteed Bounds for Posterior inference) gubpi-tool.github.io

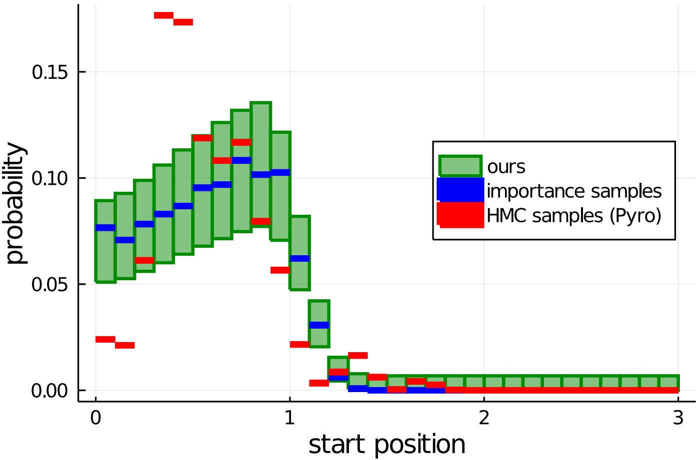
Pedestrian example:



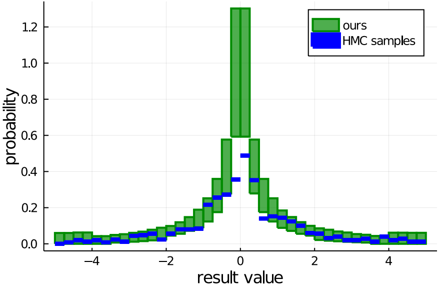
Empirical Evaluation

- Implementation: GuBPI (Guaranteed Bounds for Posterior inference) gubpi-tool.github.io

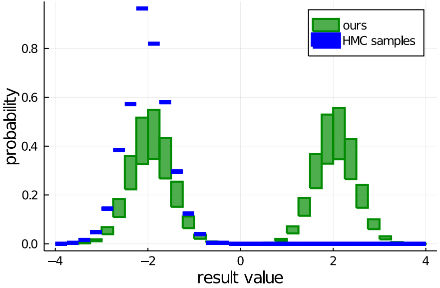
Pedestrian example:



Examples that Trip Up MCMC!



Neal's funnel



Mixture model

Comparison with Previous Work

Sankaranarayanan et al. (PLDI2013)

- framework for bounding probabilities of events definable by score-free programs
- ours is more general, hence usually slower, but often finds tighter bounds

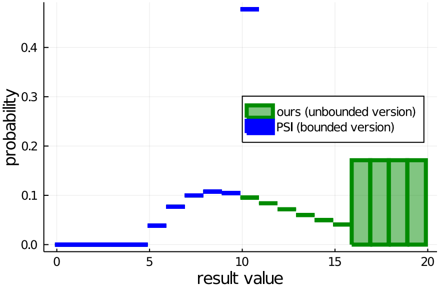
Comparison with Previous Work

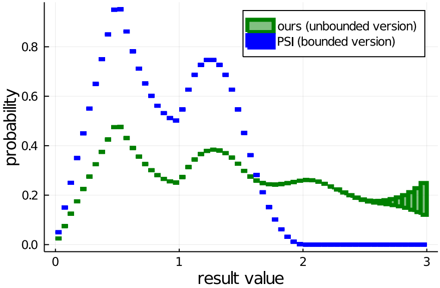
Sankaranarayanan et al. (PLDI2013)

- framework for bounding probabilities of events definable by score-free programs
- ours is more general, hence usually slower, but often finds tighter bounds

PSI solver (CAV2016)

- consistency check: benchmarks from the PSI repository
- we can handle unbounded loops, contrary to PSI. Warning: Artificially bounding recursive programs can yield different (hence wrong) posterior distribution!





Also in our PLDI22 paper[1]

- **Constraint-based interval type system:** approximates unbounded loops and recursion *soundly*
- **Symbolic execution & linear programming:** optimization for linear guards
- **Comparison with statistical validation methods:** simulation-based calibration

[1] Raven Beutner, C.-H. Luke Ong, Fabian Zaiser: Guaranteed bounds for posterior inference in universal probabilistic programming. PLDI 2022: 536-551

Also in our PLDI22 paper[1]

- **Constraint-based interval type system:** approximates unbounded loops and recursion *soundly*
- **Symbolic execution & linear programming:** optimization for linear guards
- **Comparison with statistical validation methods:** simulation-based calibration

Limitations: GuBPi struggles if

- program has lots of branching: path explosion problem
- model is high-dimensional (i.e. has many `sample`s)

[1] Raven Beutner, C.-H. Luke Ong, Fabian Zaiser: Guaranteed bounds for posterior inference in universal probabilistic programming. PLDI 2022: 536-551

Guaranteed Bounds for Posterior Inference in Universal Probabilistic Programming

Raven Beutner Luke Ong Fabian Zaiser

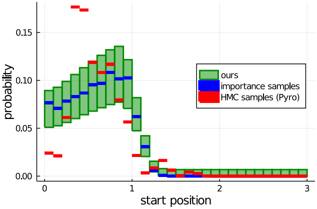
. . . are a **middle ground** between *approximate* and *exact*:

- guaranteed correct (vs. approximate inference)
- supports many language features (vs. exact inference)

Theory: soundness & completeness

Practice:

- detect issues with inference results
- competitive on existing benchmarks
- guaranteed partial correctness specifications for programs that other tools cannot handle



Future work

- better heuristics for finding a "good" set of interval traces $\mathcal{T}$
- basis for a new approximate inference algorithm.