Sound Abstraction and Decomposition of Probabilistic Programs

Steven Holtzen and Guy Van den Broeck and Todd Millstein

University of California, Los Angeles

{sholtzen,guyvdb,todd}@cs.ucla.edu

Introduction: What are Probabilistic Programs?

- Probabilistic programs are programs that contain random variables:

```
x = flip(1/2);
y = flip(1/8);
z = x ∨ y;
```

- Defines a probability distribution over program states
- Goal: To perform probabilistic inference, i.e. compute

$$\Pr(z)$$

Motivation

- Probabilistic programs are *naturally compositional*
  - Easy to build large complex models out of simple small ones
  - A key part of their expressive power and usefulness
  - Ex: Programs that are both continuous and discrete, combinations of different families of probability models
- Problem: Inference algorithms are *not compositional*
  - Treat program as black box

Edward

PYRO

STAN

  - Do not exploit program structure
  - Many simple programs combine to one very hard program

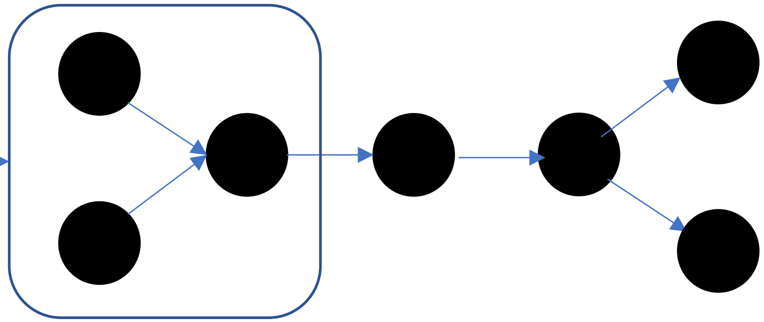
Goal

- Our goal: to automatically *decompose probabilistic programs*
- Inference becomes compositional
  - Perform inference on each sub-program
  - Combine to yield results on entire program
- Exploit program structure
  - Build complex programs out of simple parts

Key Idea: Decomposition by Abstraction

- Observation: In general, decomposition is driven by *abstraction*
- **Example:** Decomposition in graphical models

First reason about this block



- Graph *abstracts away* irrelevant details of underlying distribution
- Inference algorithms driven by graph structure, exploit sparsity to decompose the inference task

Research Questions

1. What is an appropriate notion of abstraction for probabilistic programs?
2. How can this abstraction be used to decompose inference?
3. Can we automatically produce such abstractions?
4. Can this abstraction procedure improve the performance of inference algorithms in practice?

Probabilistic Predicate Abstraction

- Q: What is an appropriate notion of abstraction for probabilistic programs?
- A: A *probabilistic predicate abstraction*, captures the probability distribution on predicates on the original program

$$
\begin{aligned}
&x \leftarrow \texttt{discrete\_dist()};\\
&y \leftarrow \texttt{continuous\_dist()};\\
&z \leftarrow x * \texttt{floor}(y);
\end{aligned}
$$

C

Abstract

$$
\begin{aligned}
&\{x = 0\} \leftarrow \texttt{flip}(\theta_{x=0});\\
&\{0 \leq y < 1\} \leftarrow \texttt{flip}(\theta_{0 \leq y < 1});\\
&\{z = 0\} \leftarrow \{x = 0\} \vee \{0 \leq y < 1\};
\end{aligned}
$$

A

Predicates, true/false statements about the program

- Goal: Compute $\Pr(z = 0)$

Probabilistic Predicate Abstraction

- Q: How do we relate this abstraction to the original program for the purpose of inference?
- A: Choose parameters of the abstraction to match the distribution in the original program (distributional soundness)

$x \leftarrow \texttt{discrete\_dist();}$
$y \leftarrow \texttt{continuous\_dist();}$
$z \leftarrow x * \texttt{floor}(y)\texttt{;}$

C

Abstract

$\{x = 0\} \leftarrow \texttt{flip}(\theta_{x=0})\texttt{;}$
$\{0 \leq y < 1\} \leftarrow \texttt{flip}(\theta_{0 \leq y < 1})\texttt{;}$
$\{z = 0\} \leftarrow \{x = 0\} \vee \{0 \leq y < 1\}\texttt{;}$

A

Exact inference

$$\theta_{x=0} = \Pr(\texttt{discrete\_dist()} = 0)$$

$$\theta_{0 \leq y < 1} = \Pr(0 \leq \texttt{continuous\_dist()} < 1)$$

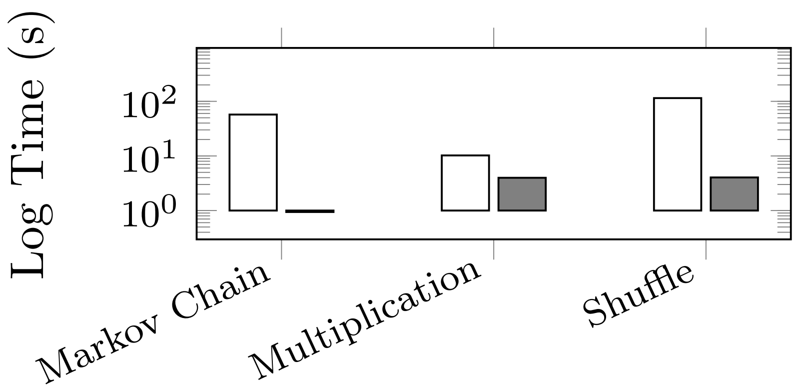
Hamiltonian Monte-Carlo

Producing Abstractions

- Q: Can we automatically produce such abstractions?
- A: Yes!
  - We show it is always possible, provide an algorithm
  - Based on predicate abstraction, well-known technique in the program analysis community

Experiments: Is this actually useful?

- Exact inference using the Psi probabilistic programming system (Gehr et al. 2016)
- Orders of magnitude improvements by using abstractions
- Recover well-known exact inference techniques (e.g. join tree)

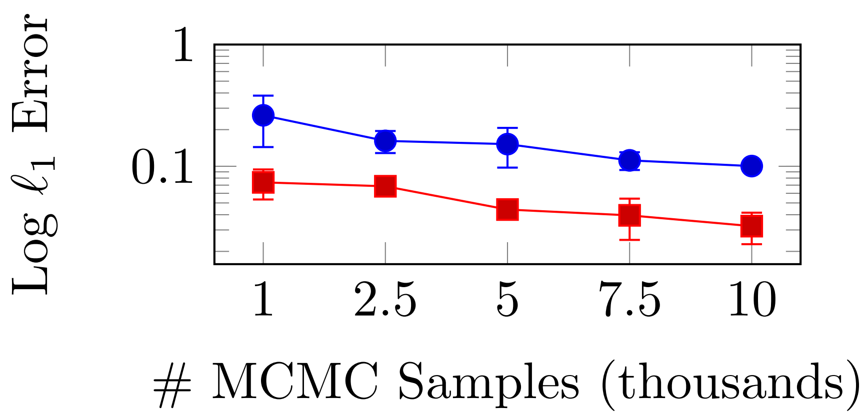


Gray bar: decomposition via abstraction
White bar: no abstraction

Experiments: Is this actually useful?

- Approximate inference using MCMC and a fixed sample budget

- Faster convergence rate for MCMC



Blue line: no abstraction
Red line: decomposition via abstraction

Conclusion

- It is possible to build abstractions of probabilistic programs
- It is helpful for improving inference in practice, can be applied to existing probabilistic programming systems
- Now, we care about
  - Automatically finding abstractions
  - Generalizing to wider family of programs

Questions?

Poster #24

Extra slides

Running Example

- Input Program
- Goal: to compute $\Pr(z = 0)$

$x \leftarrow$`discrete_dist();`
$y \leftarrow$`continuous_dist();`
$z \leftarrow x *$`floor`(y);

- This is hard for existing probabilistic programming systems
  - Mixture of continuous and discrete sub-programs
  - Non-differentiable, high-dimensional
- Yet, the program is very structured

$$(z = 0) \iff [(x = 0) \vee (0 \leq y < 1)]$$

Abstractions of Probabilistic Programs

$$x \leftarrow \texttt{discrete\_dist()};$$
$$y \leftarrow \texttt{continuous\_dist()};$$
$$z \leftarrow x * \texttt{floor}(y);$$

- Input Program
- Goal: to compute $\Pr(z = 0)$
- Observation: we know $(z = 0) \iff [(x = 0) \vee (0 \leq y < 1)]$

- Key idea: model distribution on some collection of *predicates*

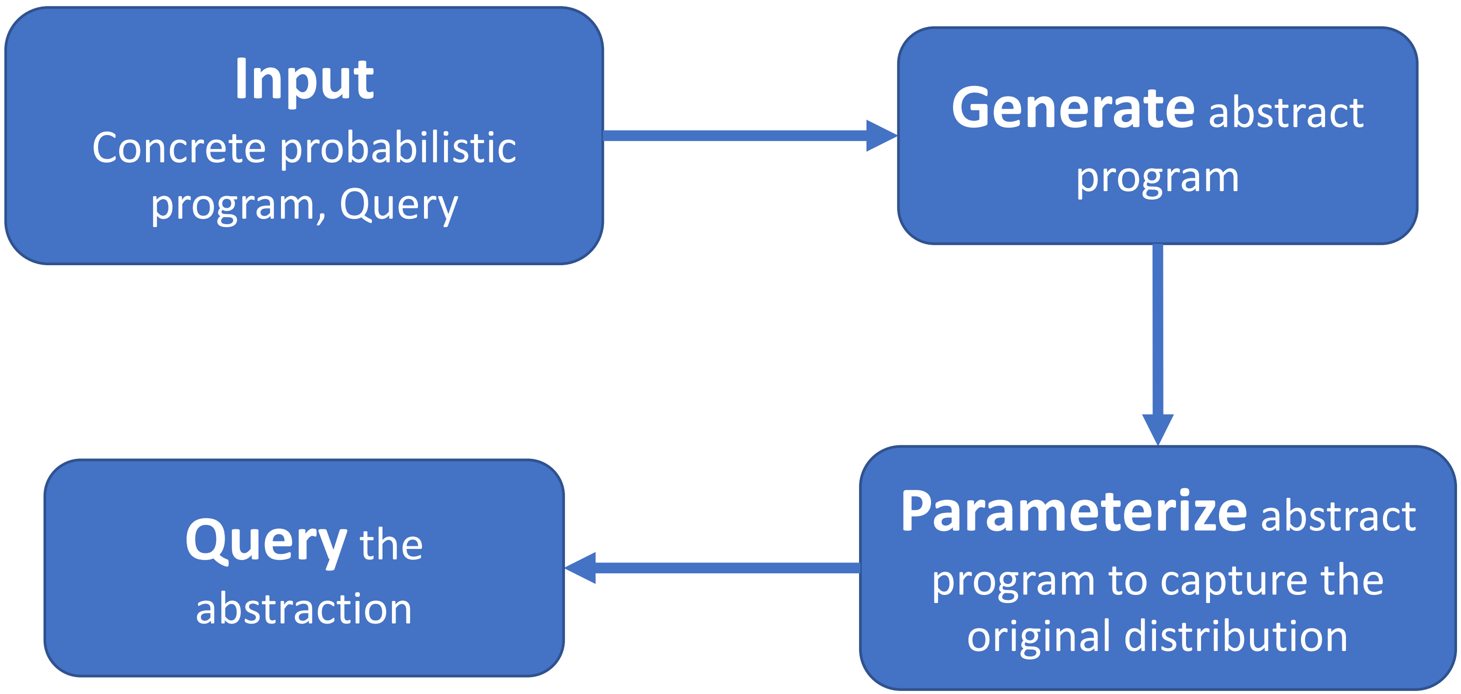
$$\theta_{x=0} = \Pr(\texttt{discrete\_dist()} = 0)$$ ← Exact inference

$$\theta_{0 \leq y < 1} = \Pr(0 \leq \texttt{continuous\_dist()} < 1)$$ ← Hamiltonian Monte-Carlo

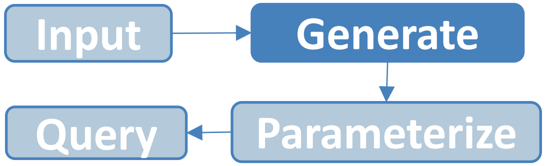
- From these random variables, we can answer the original query, and have decomposed the program

The High-Level Idea

- Decomposition via abstraction



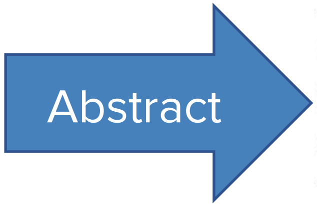
Predicate Abstraction



- Input: Probabilistic program, fixed set of predicates
- Output: Abstract probabilistic program which captures behavior on those predicates

$$x \leftarrow \texttt{discrete\_dist}();$$
$$y \leftarrow \texttt{continuous\_dist}();$$
$$z \leftarrow x * \texttt{floor}(y);$$

C

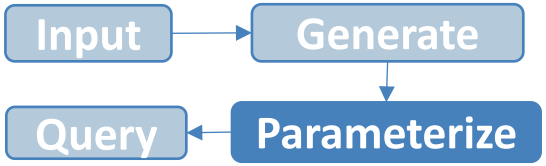


$$\{x = 0\} \leftarrow \texttt{flip}(\theta_{x=0});$$
$$\{0 \le y < 1\} \leftarrow \texttt{flip}(\theta_{0 \le y < 1});$$
$$\{z = 0\} \leftarrow \{x = 0\} \lor \{0 \le y < 1\};$$

A

- Abstraction still not useful for inference: distribution needs to *be connected* to the original program

Parameterization and Decomposition



- Choose parameters for abstraction *so that it mirrors the distribution on the concrete program*

$$\begin{array}{l} x \leftarrow \texttt{discrete\_dist}(); \\ y \leftarrow \texttt{continuous\_dist}(); \\ z \leftarrow x * \texttt{floor}(y); \end{array}$$

C

Abstract

$$\begin{array}{l} \{x = 0\} \leftarrow \texttt{flip}(\theta_{x=0}); \\ \{0 \leq y < 1\} \leftarrow \texttt{flip}(\theta_{0 \leq y < 1}); \\ \{z = 0\} \leftarrow \{x = 0\} \lor \{0 \leq y < 1\}; \end{array}$$

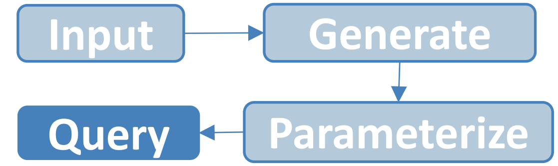
A

- Compute *sub-queries* on the original program

$$\theta_{x=0} = \Pr(\texttt{discrete\_dist}() = 0)$$ Exact inference

$$\theta_{0 \leq y < 1} = \Pr(0 \leq \texttt{continuous\_dist}() < 1)$$ Hamiltonian Monte-Carlo

- *Decomposition*: separates reasoning about the two sub-programs



Query the abstraction

- Once abstraction is properly parameterized we can query it to answer questions about the original program

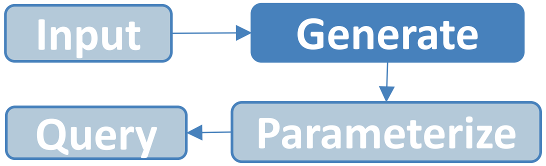
```
x ← discrete_dist();
y ← continuous_dist();
z ← x * floor(y);
```

C

$\{x = 0\} \leftarrow \texttt{flip}(\theta_{x=0});$
$\{0 \leq y < 1\} \leftarrow \texttt{flip}(\theta_{0 \leq y < 1});$
$\{z = 0\} \leftarrow \{x = 0\} \vee \{0 \leq y < 1\};$

A

- Structure of abstraction tells us how to combine sub-queries to answer the original query



Predicate Abstraction

```
x ←discrete_dist();
y ←continuous_dist();
z ← x * floor(y);
```

- **Key idea**: generate a simpler probabilistic program which only manipulates predicates
- Preserve behavior of the original program *on those predicates*
- Example: exploiting prop

$z = 0$ if and only if

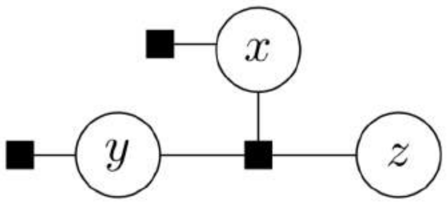
An old and effective idea from deterministic program analysis (Graf & Saidi, 1997; Ball et al., 2001)

Existing Work: Graphical Model Abstractions of Probabilistic Programs

- Idea: *abstract* the program into a probabilistic graphical model, like a factor graph

```
x ← discrete_dist();
y ← continuous_dist();
z ← x * floor(y);
```

Abstract



1. **Semantic benefits**: compactly represent independences, conditi
2. **Computational benefit** algorithms on the grap

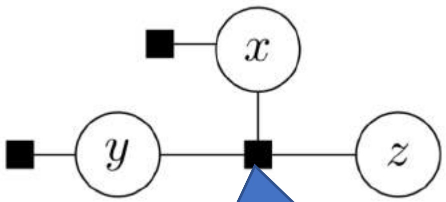
Explored in existing systems:
- Figaro (Pfeffer 2009)
- Infer.NET (Minka et al. 2014)
- Factorie (McCallum et al. 2009)

Graph-Based Abstractions are Insufficient

- Idea: *abstract* the program into a probabilistic graphical model, like a factor graph

```
x ← discrete_dist();
y ← continuous_dist();
z ← x * floor(y);
```

Abstract



Treats factors as a black box, loses structure of multiplication

- Does this abstraction make inference
  - Sometimes, but not always
  - In this case, no: There are no conditional independences in the graph if we want to compute $\Pr(z = 0)$

The Value of Graph-Based Abstraction

- Why abstract? To capture key properties and ignore irrelevant details.
- *Semantically encode* useful properties: independences, conditional probabilities, etc.
- *Computationally* reason at the level of the graph

Joint Distribution

Burglar(B)	Earthquake(E)	Alarm(A)	Probability
T	T	T	θ_1
T	T	F	θ_2
T	F	T	θ_3
…	…	…	…

Bayesian Network