Scaling Probabilistic Programming Up

Steven Holtzen

IFIP Working Group 2.8

May 2025

Northeastern University

s.holtzen@northeastern.edu

Overview

Talk goal: *to explain and motivate the language design challenges in probabilistic programming languages.*

1. Introduction and overview of probabilistic programming languages (PPLs): programming languages with probabilistic semantics
2. `Roulette`: a fast and expressive PPL based on Rosette
3. Conclusion: perspectives on making PPLs more widespread and usable

Two arguments for probabilistic programming languages

PL

1. It is useful to verify programs.
2. Many kinds of programs have inherent probabilistic uncertainty.
3. We need programming languages with probabilistic semantics to give a semantics to these programs.

AI

1. We want to create agents that reason rationally about the world.
2. To do this, we need a language for describing the world to a computer.
3. The world is too complicated to describe without probabilities.
4. We need programming languages with probabilistic semantics.

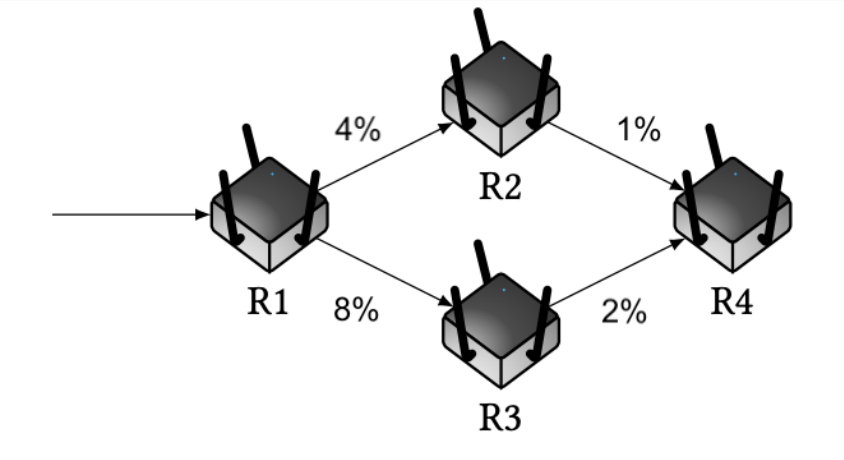
What are probabilistic programming languages (PPLs)?

Programming with `Roulette` [16]: a new probabilistic programming language in Racket based on Rosette

```
#lang roulette/example/disrupt
> (flip 0.4)
(pmf | #t ↦ 0.4 | #f ↦ 0.6)
```

- `(flip 0.4)` *introduces* randomness, `#t` with probability 0.4 and `#f` with probability 0.6.
- Outputs a probability mass function (PMF), associates values to probabilities

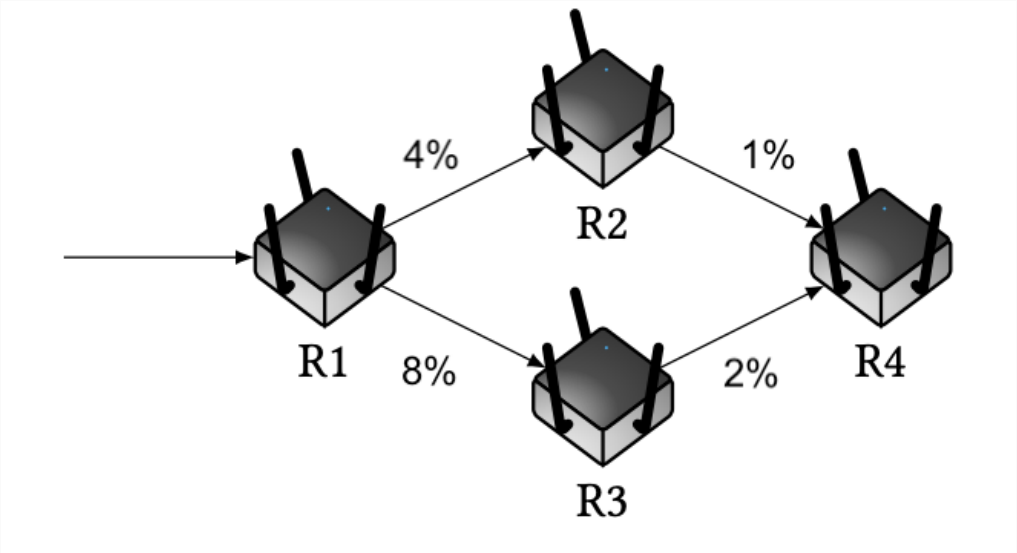
Programming with probabilities: network reachability



Goal: Compute probability an incoming packet reaches R4:

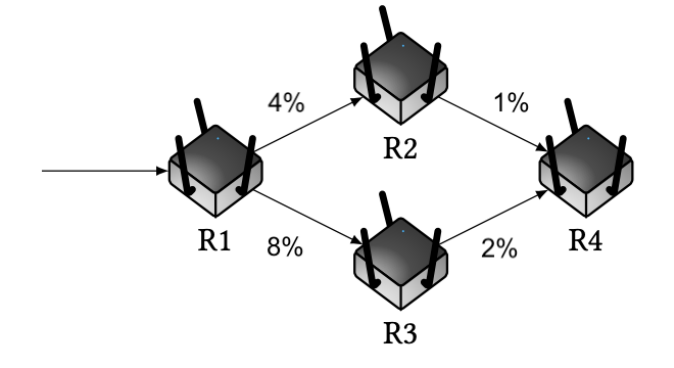
- Packet forwards from R1 uniformly at random
- Each edge fails with the annotated probability

Programming with probabilities: network reachability




```
> (if (flip 0.5)  ; if true, forward to R2
    (and (flip 0.96) (flip 0.99))
    (and (flip 0.92) (flip 0.98)))
(pmf | #t ↦ 0.926 | #f ↦ 0.074)
```

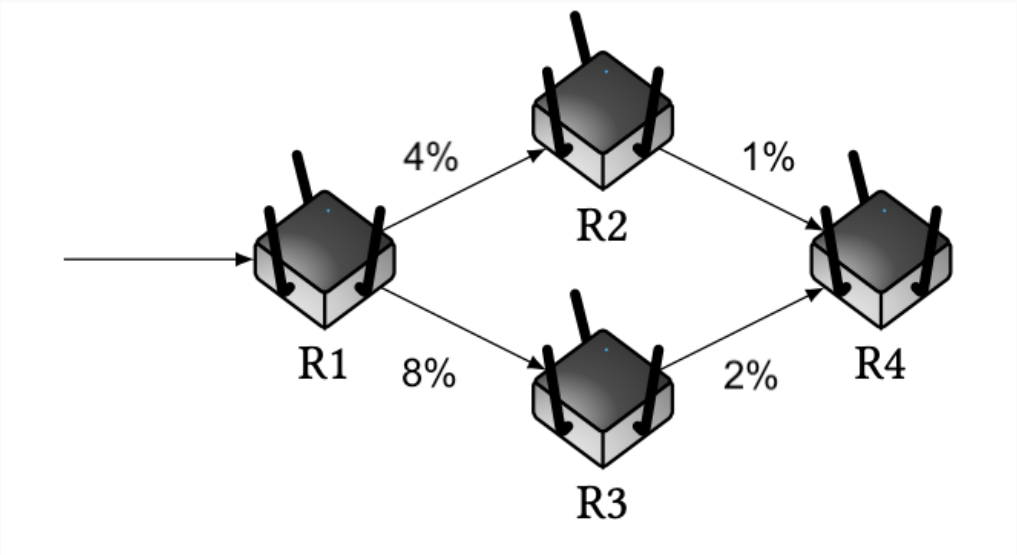
Programming with probabilities: network diagnosis



Now suppose we want to perform *network diagnosis*.

We **observe** a packet does not successfully reach R4. What is the probability that the R1 – R4 link failed?

Programming with probabilities: network diagnosis




```
> (define r1->r2 (flip 0.96))
> (define reaches (if (flip 0.5)  ; if true, forward to R2
    (and r1->r2 (flip 0.99))
    (and (flip 0.92) (flip 0.98))))
> (observe! (not reaches))
> r1->r2
(pmf | #t ↦ 0.703135 | #f ↦ 0.296865)
```

`Roulette` is quite expressive: Demo #1

Add 1 to each element of the list with probability 1/2

```
> (define my-list '(1 2 3))
> (map (lambda (x)
         (if (flip 0.5)
             x
             (+ x 1)))
       my-list)

(pmf | '(2 2 4) ↦ 0.125
     | '(2 3 3) ↦ 0.125
     | '(2 3 4) ↦ 0.125
     | '(1 3 4) ↦ 0.125
     ...
```

`Roulette` is quite expressive: Demo #2

Compute probability that sum of list is even

```
(define my-list '(1 2 3))

(define (add1-randomly l)
  (map (lambda (x)
         (if (flip 0.5)
             x
             (+ x 1)))
       l))

(define (sum-list l)
  (foldr (lambda (x acc) (+ x acc))
         0 l))

> (equal? (modulo (sum-list (add1-randomly my-list)) 2)
        0)

(pmf | #t ↦ 0.5 | #f ↦ 0.5)
```

Some applications of PPLs, real and imagined

- Teaching probability
- Network verification [9, 19]
- Medical diagnosis
- Scientific discovery
- Differential privacy [18, 1, 6]
- Fraud detection
- Verifying cryptography
- Randomized algorithms

There is *high demand* for effective PPLs!

- (Verified) machine learning
- Markov decision processes [12]
- Pharmaceuticals research [17]
- Social sciences [10]
- Content generation for games [15]
- ...

The central challenge: scalability

Probabilistic inference (evaluating semantics) is computationally hard even for very restricted languages.

$$e ::= \texttt{(if g thn els)} \mid (\texttt{flip}\ \theta) \mid \texttt{(let id e1 e2)} \mid \texttt{id}$$

Even for this restricted language, inference is #NP-hard! Fairly easy reduction to SAT.

Scaling PPLs Up: Inference

Hope: *Inference is worst-case intractable, but it is not always intractable: real programs have structure.*

1. Identify classes of programs for which inference is tractable.
   - Type systems, static analysis, programmer discipline, etc.
2. Develop inference algorithms that exploit this tractability.
3. Develop systems that deploy the correct inference algorithm to problems with the right kinds of structure.

Scaling Inference:

Design and Implementation of `Roulette`

Inference landscape: Enumeration

- Simplest idea: exhaustively enumerate all the possible values for each random variable
- Easily implemented in Haskell using the probability (or Giry) monad

```
newtype Dist a = Dist [(a, Double)]

dflip :: Double -> Dist Bool
dflip f = Dist([(True, f), (False, 1 - f)])

my_val :: Dist Bool
my_val =
  do
    x <- (dflip 0.5);
    y <- (dflip 0.5);
```

Expressive (all of Haskell!)

but slooooooow!

```
Dist [(True,0.25),(False,0.25),(False,0.25),(False,0.25)]
```

Inference landscape: Direct Sampling

- Also simple idea: when you encounter a `flip`, sample a value from the Bernoulli probability distribution. Run program many times, take average
- Answers are now *approximate*. Scales better (sometimes!), weaker guarantees
- Definitely the most common approach to inference

Expressive, exact, efficient

It is very hard to make a PPL that is:

1. *Expressive*: Programming in the language should feel normal. Few weird restrictions.
2. *Exact*: The programmer should get exact deterministic answers to their queries
3. *Efficient*: Fast enough to be practically useful. As fast as specialized solvers?

This is what we want `Roulette` to do.

How `Roulette` happened

"Lecture 6: Dice is a probabilistic programming language that works by compiling programs into logical formulae and performing weighted model counting..."

Cameron Moy: Hey, that sounds a lot like what Rosette does!

Tractability–Expressivity Tradeoff

Expressive Language $\longleftrightarrow$ **Tractable Language**

- Inference is *undecidable*
- Continuous distributions
- General recursion
- ... other features

- Middle-ground: decidable inference, but high complexity
- Sure termination
- Typically discrete
- ... more?

- **Tractable probabilistic models** (TPMs): Inference is *polytime in size*.
- Highly restrictive: an *assembly language* (looks like ANF)
- Examples: binary decision diagrams (BDDs), generating functions, ...

Approach: Inference-via-Compilation

- Separate inference implementation from language design
  - High-level language has ergonomic features, pleasant to program in
  - Target language captures *problem-specific structure*
  - Compiler may be expensive, inference cost is amortized
- Develop new tractable target languages that enable new higher-level language features
- Increasing number of examples: Dice [11, 8], ProbLog [7], SPPL [7], generating functions [13], ...

`Roulette`: an expressive, exact, and efficient PPL

Probabilistic Subset of Racket

- Discrete random variables
- General recursion (sure termination)
- Macros
- Mutable state
- General datatypes (recursive, higher-order, ...)
- Interop with Racket

Tractable PPL

Knowledge compilation

- High-performance exact inference: essentially as fast as state-of-the-art exact inference for Bayesian networks
- Based on Rosette [21], implemented as a `#lang`

`Roulette` by Example: Symbolic Unions

Step 1: Run a `Roulette` program under abstract semantics ($\Downarrow$) to a symbolic union and weight map.

$$\left[f_1 \vee f_2 : \texttt{\#t}, \neg(f_1 \vee f_2) : \texttt{\#f}\right]$$

```
(define f1 (flip 0.4))
(define f2 (flip 0.7))
(or f1 f2)
```

$\Downarrow$

Literal ℓ	$w(\ell)$
f_1	0.4
$\neg f_1$	0.6
f_2	0.7
$\neg f_2$	0.3

- A *symbolic union* is a list of Racket values guarded by logical formulae [21].
- Racket operations are *lifted* to operate on symbolic unions.

`Roulette` by Example: Symbolic Unions

Another example:

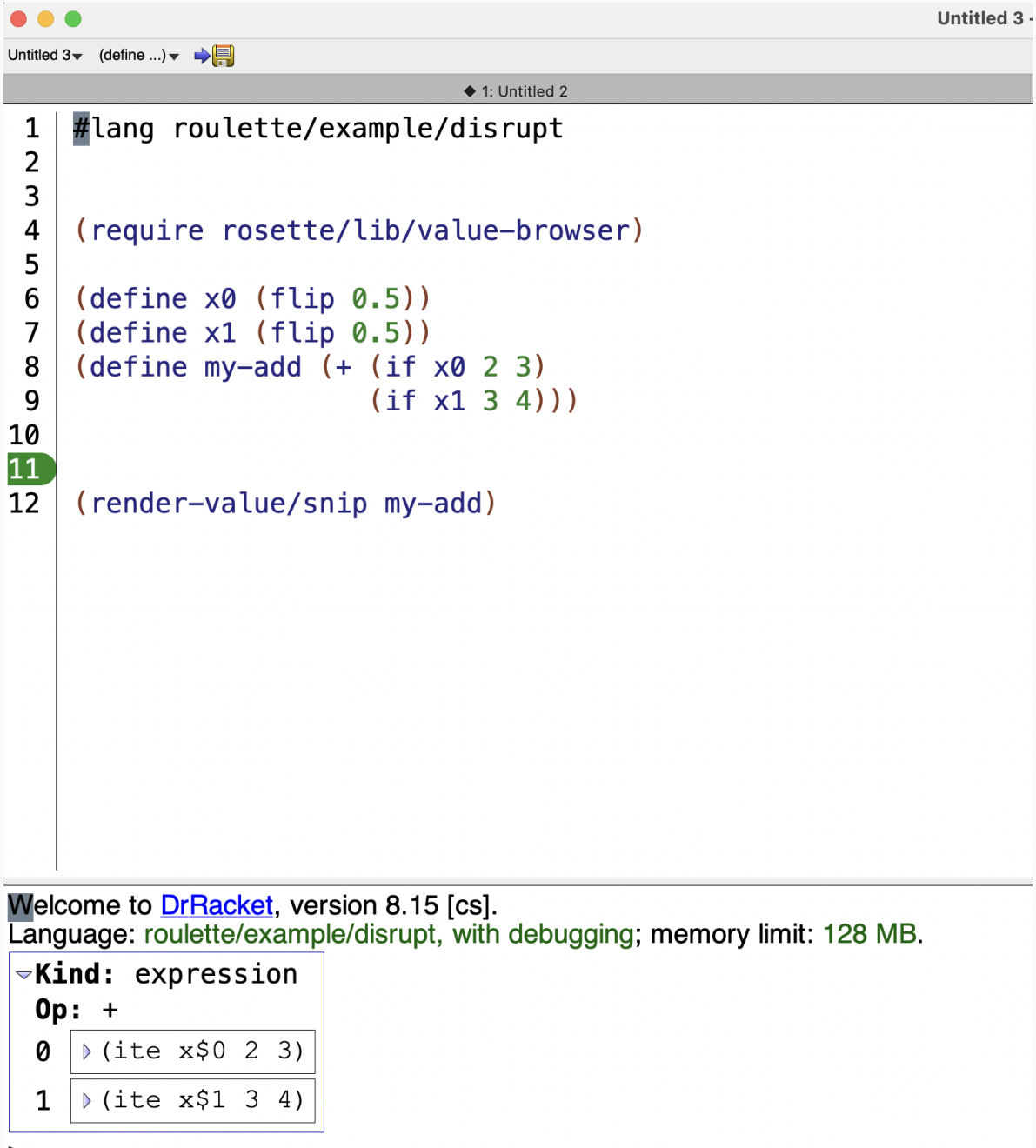
```
(define x0 (flip 0.5))
(define x1 (flip 0.5))
(+ (if x0 2 3)
   (if x1 3 4))
```

$\Downarrow\!\Downarrow$

$$\Big[x_0 \wedge x_1 : 5,\ (\neg x_0 \wedge x_1) \vee (x_0 \wedge \neg x_1) : 6,\ (\neg x_0 \wedge \neg x_1) : 7\Big]$$

Literal ℓ	$w(\ell)$
x_1	0.5
$\neg x_1$	0.5
x_2	0.5
$\neg x_2$	0.5

Roulette by Example: Symbolic Unions



Snippet of full abstract semantics

$$\hat{\rho} \vdash (e, \hat{\sigma}, w) \Downarrow\!\Downarrow (\hat{v}, \hat{\sigma}', w', \psi)$$

- In (abstract) environment $\hat{\rho}$
- The program e with store $\hat{\sigma}$ and weight map w
- Runs to value $\hat{v}$, new store $\hat{\sigma}'$, new weight map w', and asserting formula ψ (for conditioning)

Snippet of full abstract semantics

Simplified `flip` rule with constant parameter r:

$$\frac{\alpha \text{ fresh}}{\hat{\rho} \vdash (\texttt{(flip r)}, \hat{\sigma}, w) \Downarrow ([\alpha : \texttt{\#t}, \neg\alpha : \texttt{\#f}], \hat{\sigma}, w[\alpha \mapsto \texttt{r}, \neg\alpha \mapsto 1 - r], \texttt{T})}$$

$$\hat{\rho} \vdash (e, \hat{\sigma}, w) \Downarrow (\hat{v}, \hat{\sigma}', w', \psi)$$

Snippet of full abstract semantics

$$\textsc{If}\quad \frac{\widehat{\rho}(x) = [\varphi_1 : \texttt{true}, \varphi_2 : \texttt{false}] \qquad \widehat{\rho} \vdash (e_1, \widehat{\sigma}) \Downarrow (\widehat{v}_1, \widehat{\sigma}_1, \psi_1) \qquad \widehat{\rho} \vdash (e_2, \widehat{\sigma}) \Downarrow (\widehat{v}_2, \widehat{\sigma}_2, \psi_2)}{\widehat{\rho} \vdash (\texttt{if}\ x\ e_1\ e_2, \widehat{\sigma}) \Downarrow ([\varphi_1 : \widehat{v}_1, \varphi_2 : \widehat{v}_2], [\varphi_1 : \widehat{\sigma}_1, \varphi_2 : \widehat{\sigma}_2], (\varphi_1 \wedge \psi_1) \vee (\varphi_2 \wedge \psi_2))}$$

$$\hat{\rho} \vdash (e, \hat{\sigma}, w) \Downarrow (\hat{v}, \hat{\sigma}', w', \psi)$$

Computing probabilities

- The *weight of a model* $w(m)$ is the product of the weights of each literal in the model: $w(m) = \prod_{\ell \in m} w(\ell)$
- The *weighted model count* $\texttt{WMC}(\varphi, w)$ is the weighted sum of the models of φ:

$$\texttt{WMC}(\varphi, w) = \sum_{m \models \varphi} w(m)$$

Computing probabilities

Theorem 5.1, very informal: weighted model counting of the symbolic union computes the correct probabilities.

$$\left[f_1 \vee f_2 : \texttt{\#t}, \neg(f_1 \vee f_2) : \texttt{\#f}\right]$$

```
(define f1 (flip 0.4))
(define f2 (flip 0.7))
(or f1 f2)
```

$\Downarrow$

Literal ℓ	$w(\ell)$
f_1	0.4
$\neg f_1$	0.6
f_2	0.7
$\neg f_2$	0.3

To compute the probability that the program evaluates to `#t`, compute the `WMC` of the guard for that value:

$$\texttt{WMC}(f_1 \vee f_2) = 0.4 \times 0.7 + 0.4 \times 0.3 + 0.6 \times 0.7 = 0.82$$

Scalable inference

- We've reduced `Roulette` inference to performing `WMC`
- Now we can deploy specialized *weighted model counting tools*!
- Rich catalog of tools for this, similar to SAT!

Historical context: inference via weighted model counting is a state-of-the-art approach for exact inference in discrete Bayesian networks, goes back to Chavira and Darwiche [4]

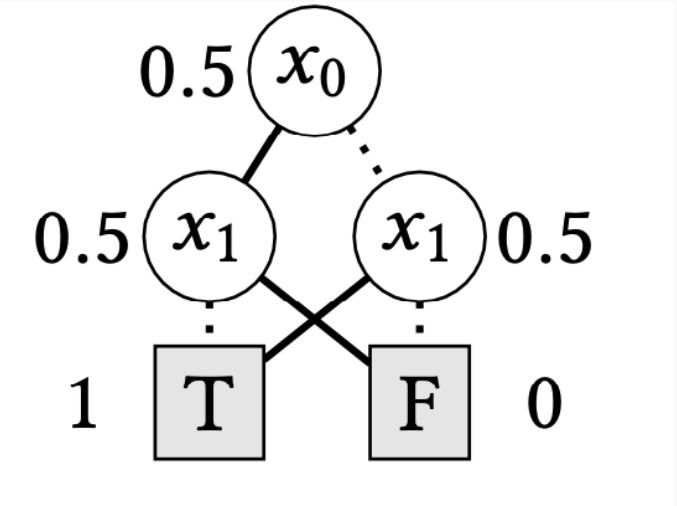
Knowledge compilation

- **Step 2**: compile formulae to binary decision diagrams (BDDs), which support linear-time weighted model counting

$$\Big[x_0 \wedge x_1 : 5,\ (\neg x_0 \wedge x_1) \vee (x_0 \wedge \neg x_1) : 6,\ (\neg x_0 \wedge \neg x_1) : 7\Big]$$

$\rightsquigarrow$ Compile guard for 6

Literal ℓ	$w(\ell)$
x_1	0.5
$\neg x_1$	0.5



The `RSDD` knowledge compilation library: `https://github.com/neuppl/rsdd`

`Roulette` is fast

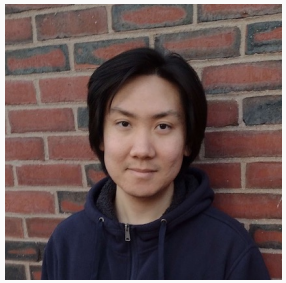
Slice of benchmarks: some challenging large Bayesian networks (many thousands of `flips`!)

	Roulette		`Dice (2024)`		`Dice (2020)`	
Benchmark	BDD Size	Time (ms)	BDD Size	Time (ms)	BDD Size	Time (ms)
cancer	**13**	**2**	15	49	28	19
survey	**46**	**4**	48	29	73	18
alarm	981	42	**672**	308	1366	**30**
insurance	75594	395	**44,846**	643	101047	**148**
hepar2	**1,967**	140	1969	230	3936	**32**
hailfinder	**33,211**	596	–	–	65386	**428**
pigs	**19**	255	25	417	35	**48**
water	39146	**200**	**33,226**	454	51952	16083
munin	**10,307**	2400	3704	24839	11977	**1,605**

Roulette **team**

Cameron Moy

Jack Czenszak



John M. Li

Brianna Marshall

Growing `Roulette`

Dimensions of growth

- *More language expressivity*
  - More kinds of probability distributions (continuous)
  - Queries beyond inference: optimization [5], causality
- *More inference strategies*
  - Different backends, approximate inference
  - Language interoperation
  - Combining deductive reasoning with inference
- *Beyond probabilities*
  - Weighted/quantum programming
- *Usability*
  - Generalizing symbolic profilers [3]
  - Type systems for guaranteed tractability
- *Applications*
  - Teaching a course on probability using `Roulette`
  - Property-based testing?
  - Differential privacy
  - ???

Multi-language probabilistic programming

Hope: *Programs are heterogeneous and different parts have different tractable structure we can exploit separately.*

- `MultiPPL`: language interoperation for PPLs [20]

```
1  let θ be uniform 0 1 in
2  ⟨let X be flip θ in
3   let Y be flip θ in
4   observe X ∨ Y in
5   ret X ⟩S
```

- Orange language is *sampled*, purple language is *exact* (like Roulette, but not integrated yet)
- Goal: Be able to use different inference algorithms (runtimes) program

Sam Stites

Scaling PPLs Up: Deductive Reasoning

- Sometimes (often?) automated inference can't scale! What do we do then?
- Need *proof rules* to reason about program's behavior deductively (manually).
- Eventually, this can interact with automated inference

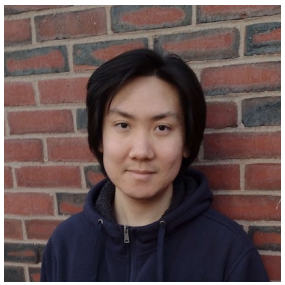
Probabilistic Separation Logics (a taste)

Hope: *Probabilistic programs have modular structure for decomposing reasoning into sub-programs.*

- *Idea*: generalize separation logic to probabilities! [2]
- Our approach: Lilac, a *measure-theoretic separation logic* for probability [14]

```
(define x (flip 0.5))
  {x ∼ bern 0.5}
(define y (flip 0.5))
  {x ∼ bern 0.5 ∗ y ∼ bern 0.5}
```

- Separating conjunction ($*$) denotes *separation of probability* stic independence



John M. Li

Conclusion

Theme: PPLs at the intersection of PL and AI

PL

- Deductive reasoning, program logics
- Formal verification
- Language interoperability
- Formal semantics
- Relative expressivity of languages

AI

- Scalable inference techniques
- Probabilistic modeling
- Tractable probabilistic models
- Broad application domains (medicine, science, robotics)

Why aren't we all using PPLs yet? Open challenges

1. **Scalability**, obviously: it's still too easy to write programs that don't scale.
   - Need more *systematic benchmarking* and perhaps open competitions, similar to SAT competition, to measure progress.
   - *Broaden applications* so we have more benchmarks. Drive language design towards problems that matter.
   - *Stronger guarantees* on inference performance and correctness. Perhaps mechanized in a proof assistant.
   - *Contentious claim*: we currently over-rely on approximate inference, not enough focus on exact inference.

Why aren't we all using PPLs yet? Open challenges

1. **Scalability**
2. **Usability**: languages are hard to use and programmers don't know how to use them.
    - *Leaky abstractions/impedence mismatch*: programmers need to know too much about how inference is implemented. Need *cost models* to help programmers understand when their programs don't scale and debug performance issues.
    - Inference diagnostics and debugging tools, profilers.
    - Goal: using a PPL in a classroom to teach undergraduates.

Links and more resources

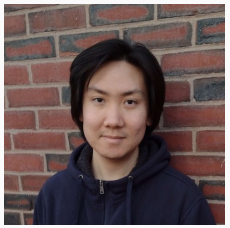
- OPLSS course on probabilistic programming: `https://www.khoury.northeastern.edu/home/sholtzen/oplss24-ppl/`
- `Roulette` source:

  `https://github.com/camoy/roulette/`
- Seminar course I taught on PPLs: `https://neuppl.github.io/CS7470-Fall23/`

Acknowledgments

Cameron Moy

Jack Czenszak



John M. Li

Brianna Marshall

Sam Stites

John Gouwar

Minsung Cho

Amal Ahmed

NSF

Sandia National Laboratories

U.S. DEPARTMENT OF ENERGY

References

[1] Gilles Barthe, Marco Gaboardi, Benjamin Grégoire, Justin Hsu, and Pierre-Yves Strub. Proving differential privacy via probabilistic couplings. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 749–758, 2016.

[2] Gilles Barthe, Justin Hsu, and Kevin Liao. A probabilistic separation logic. *Proceedings of the ACM on Programming Languages*, 4(POPL):1–30, 2019.

[3] James Bornholt and Emina Torlak. Finding code that explodes under symbolic evaluation. *Proceedings of the ACM on Programming Languages*, 2(OOPSLA):1–26, 2018.

[4] Mark Chavira and Adnan Darwiche. On probabilistic inference by weighted model counting. *Artificial Intelligence*, 172(6-7): 772–799, 2008.

[5] Minsung Cho, John Gouwar, and Steven Holtzen. Scaling optimization over uncertainty via compilation. *Proc. ACM Program. Lang.*, 9(OOPSLA1), April 2025. doi: 10.1145/3720500. URL https://doi.org/10.1145/3720500.

[6] Markus de Medeiros, Muhammad Naveed, Tancrede Lepoint, Temesghen Kahsai, Tristan Ravitch, Stefan Zetzsche, Anjali Joshi, Joseph Tassarotti, Aws Albarghouthi, and Jean-Baptiste Tristan. Verified foundations for differential privacy. *arXiv preprint arXiv:2412.01671*, 2024.

[7] Daan Fierens, Guy Van den Broeck, Joris Renkens, Dimitar Shterionov, Bernd Gutmann, Ingo Thon, Gerda Janssens, and Luc De Raedt. Inference and learning in probabilistic logic programs using weighted boolean formulas. *Theory and Practice of Logic Programming*, 15(3):358–401, 2015.

[8] Poorva Garg, Steven Holtzen, Guy Van den Broeck, and Todd Millstein. Bit blasting probabilistic programs. *Proceedings of the ACM on Programming Languages*, 8(PLDI):865–888, 2024.

[9] Timon Gehr, Sasa Misailovic, Petar Tsankov, Laurent Vanbever, Pascal Wiesmann, and Martin Vechev. Bayonet: probabilistic inference for networks. *ACM SIGPLAN Notices*, 53(4):586–602, 2018.

[10] Benjamin King Goodrich, Gregory Wawro, and Ira Katznelson. Designing quantitative historical social inquiry: an introduction to stan. In *APSA 2012 Annual Meeting Paper*, 2012.

[11] Steven Holtzen, Guy Van den Broeck, and Todd Millstein. Scaling exact inference for discrete probabilistic programs. *Proceedings of the ACM on Programming Languages*, 4 (OOPSLA):1–31, 2020.

[12] Steven Holtzen, Sebastian Junges, Marcell Vazquez-Chanlatte, Todd Millstein, Sanjit A. Seshia, and Guy Van den Broeck. Model checking finite-horizon markov chains with probabilistic inference. In *Proceedings of the 33rd International Conference on Computer-Aided Verification (CAV)*, July 2021. doi: $10.1007/978\text{-}3\text{-}030\text{-}81688\text{-}9\backslash\_27$.

[13] Lutz Klinkenberg, Kevin Batz, Benjamin Lucien Kaminski, Joost-Pieter Katoen, Joshua Moerman, and Tobias Winkler. Generating functions for probabilistic programs. In *International Symposium on Logic-Based Program Synthesis and Transformation*, pages 231–248. Springer, 2020.

[14] John M Li, Amal Ahmed, and Steven Holtzen. Lilac: a modal separation logic for conditional probability. *Proceedings of the ACM on Programming Languages*, 7(PLDI):148–171, 2023.

[15] Abdelrahman Madkour, Chris Martens, Steven Holtzen, Casper Harteveld, and Stacy Marsella. Probabilistic logic programming semantics for procedural content generation. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 19, pages 295–305, 2023.

[16] Cameron Moy, Jack Czenszak, John M. Li, Brianna Marshall, and Steven Holtzen. Roulette: A language for expressive, exact, and efficient discrete probabilistic programming. In *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, 2025. doi: 10.1145/3729334.

[17] Fanni Natanegara, Beat Neuenschwander, John W Seaman Jr, Nelson Kinnersley, Cory R Heilmann, David Ohlssen, and George Rochester. The current state of bayesian methods in medical product development: survey results and recommendations from the dia bayesian scientific working group. *Pharmaceutical statistics*, 13(1):3–12, 2014.

[18] Lisa Oakley, Steven Holtzen, and Alina Oprea. Synthesizing tight privacy and accuracy bounds via weighted model counting. In *IEEE Computer Security Foundations Symposium*, 2024.

[19] Steffen Smolka, Praveen Kumar, David M Kahn, Nate Foster, Justin Hsu, Dexter Kozen, and Alexandra Silva. Scalable verification of probabilistic networks. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 190–203, 2019.

[20] Sam Stites, John M Li, and Steven Holtzen. Multi-language probabilistic programming. *Proceedings of the ACM on Programming Languages*, 9(OOPSLA1):1239–1266, 2025.

[21] Emina Torlak and Rastislav Bodik. Growing solver-aided languages with rosette. In *Proceedings of the 2013 ACM international symposium on New ideas, new paradigms, and reflections on programming & software*, pages 135–152, 2013.

Extra slides

Bayesian learning

Flip two coins and *observe* that at least one of the two coins is `#t`.

What is the probability that the first coin is `#t`?

```
> (define coin1 (flip 0.5))
> (define coin2 (flip 0.5))
> (observe! (or coin1 coin2))
> coin1
(pmf | #t ↦ 0.666667 | #f ↦ 0.333333)
```

Possible worlds (enumeration):

- `coin1 = #t, coin2 = #t`
- `coin1 = #f, coin2 = #t`
- `coin1 = #t, coin2 = #f`
- `coin1 = #f, coin2 = #f`