Probabilistic Programming for Science and Fault Analysis

Steven Holtzen

Assistant Professor

Northeastern University

s.holtzen@northeastern.edu

https://www.khoury.northeastern.edu/home/sholtzen/

Outline

1. Motivation: Where do probabilities come from, and why do we want to automatically reason about them?

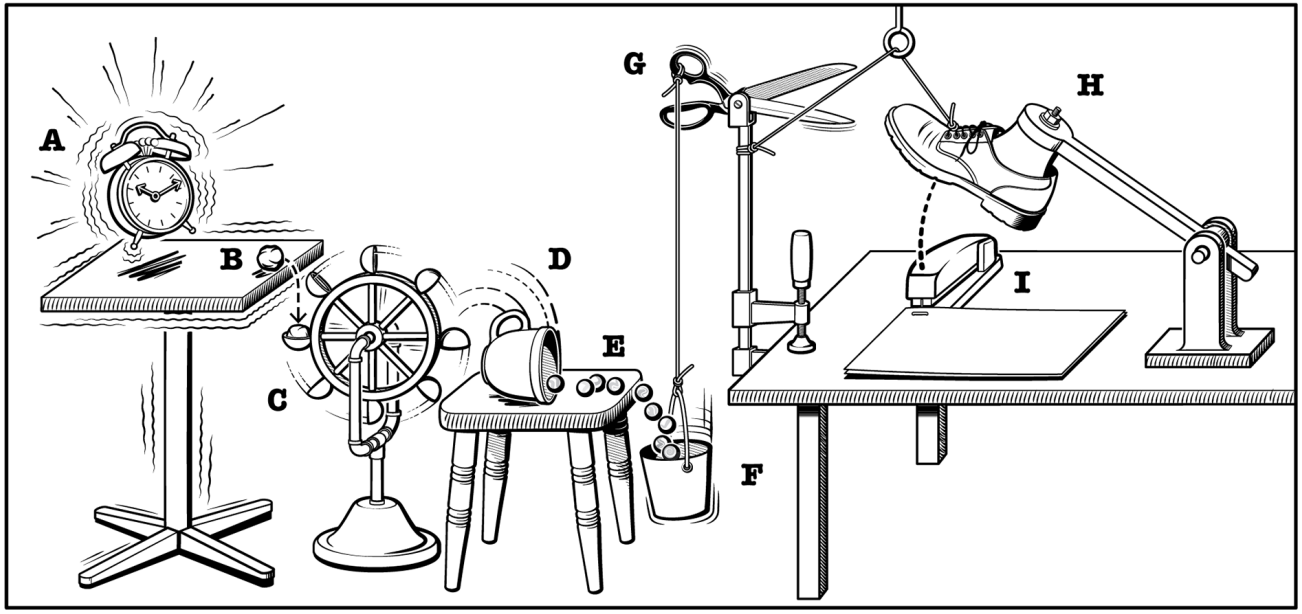
2. How *probabilistic programs* make reasoning about probabilities easier
   - Modeling and querying

3. Challenges and research directions: why doesn't everyone use probabilistic programs today?
   - Scalability and language design, case st

This is a high-level overview talk; happy to go into more detail!

Probabilistic Reasoning is Ubiquitous

1. Systems Reliability



Probability of success or failure?

Probabilistic Reasoning is Ubiquitous

2. Verifying randomized algorithms:

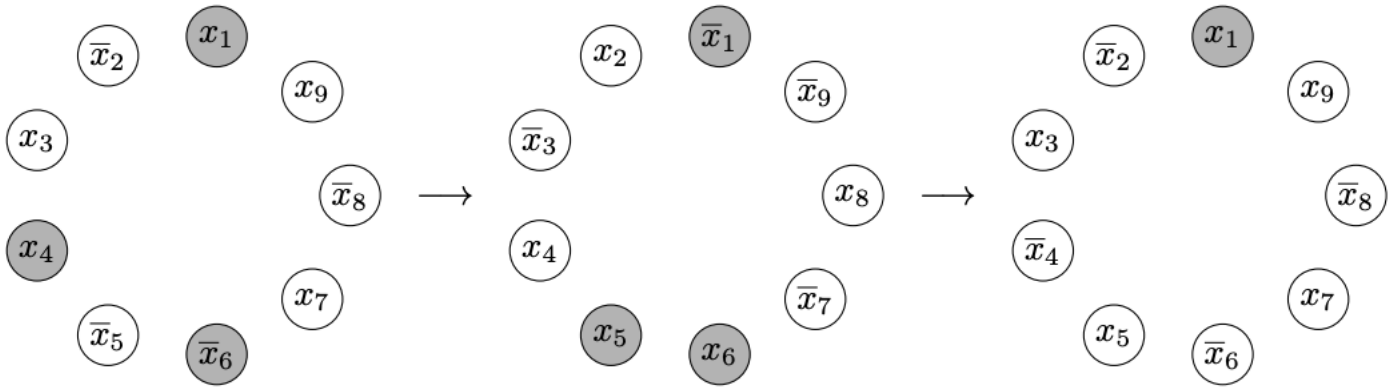


Fig. 1. Example successive configurations of Herman's ring for $N = 9$; we write x_i and $\overline{x}_i$ to denote the fact that bit x_i is 1 or 0, respectively; processes with a token are shaded grey.

Probability of termination?

[Kwiatkowska, Marta, Gethin Norman, and David Parker. "Probabilistic verification of Herman's self-stabilisation algorithm." Formal Aspects of Computing 24.4 (2012): 661-670.]

Probabilistic Reasoning is Ubiquitous

3. Reasoning about systems that integrate learned components

Inside the Self-Driving Tesla Fatal Accident

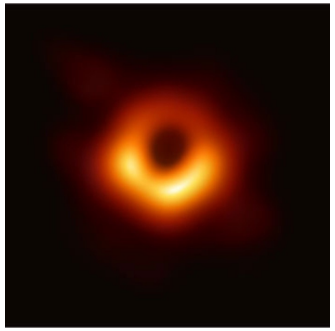
By **ANJALI SINGHVI** and **KARL RUSSELL** UPDATED July 12, 2016

The accident may have happened in part because the crash-avoidance system is designed to engage only when radar and computer vision systems agree that there is an obstacle, according to an industry executive with direct knowledge of the system.

A Typical Approach to Probability

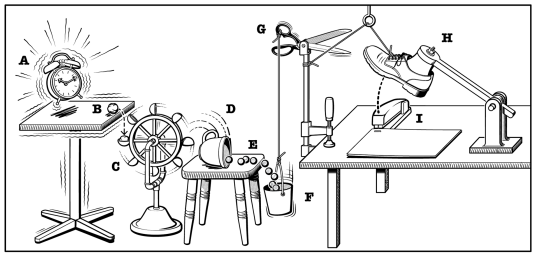
- A scientist (or other domain expert) will have some question they want to answer about a probabilistic system
  - "What is the most likely black-hole geometry gave rise to the observable black-body radiation?"
- Then, that scientist will put on their programmer hat, and make a ***custom solution for their problem***

Katie Bouman
Recently gave a keynote address at the International Conference on Probabilistic Programming

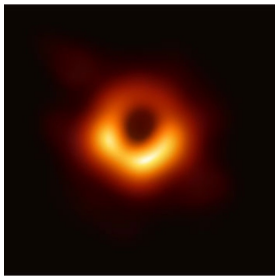


Core Challenge

- **Problem**: Proliferation of *custom solutions!*



© Vernier Software & Technology



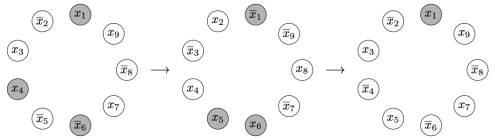
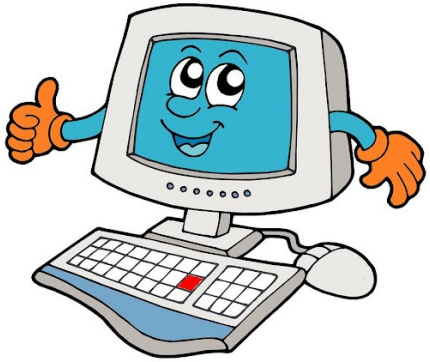


Fig. 1. Example successive configurations of Herman's ring for $N = 9$; we write x_i and $\overline{x}_i$ to denote the fact that bit x_i is 1 or 0, respectively; processes with a token are shaded grey.

- Need a way of *sharing effort across these solutions*

Probabilistic Modeling Paradigm

Model



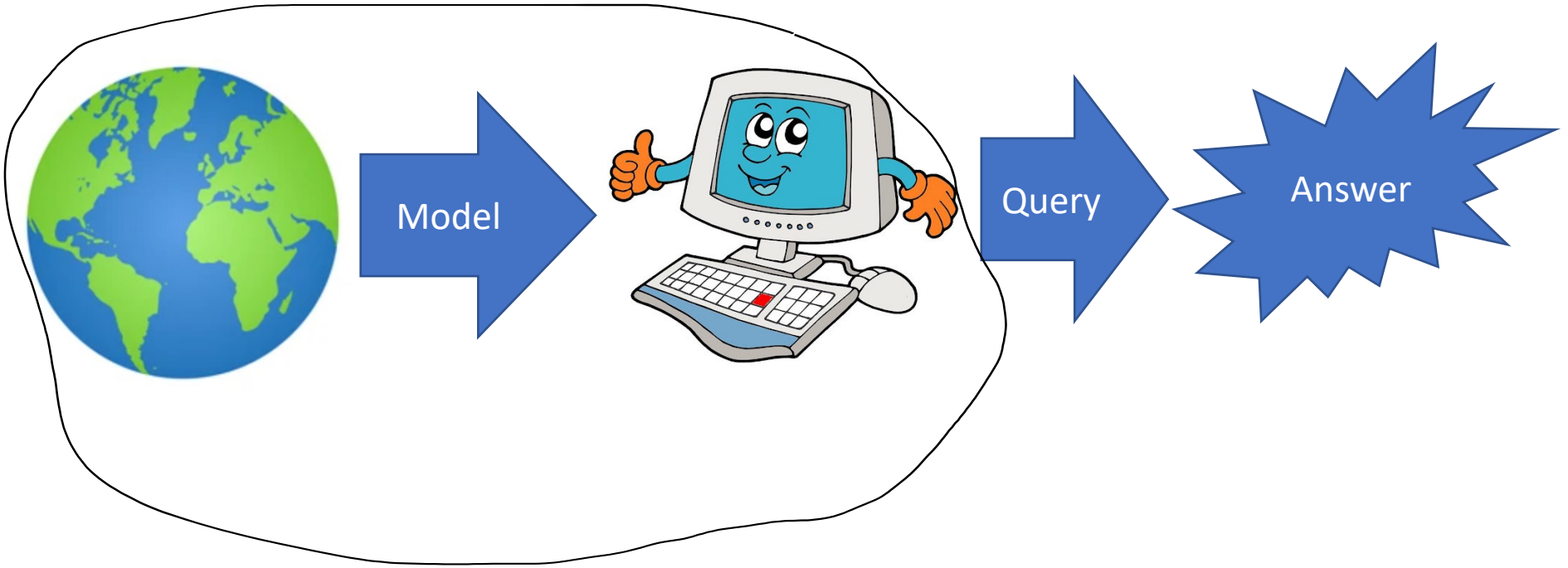
Query

Answer

- Specify the model in some ***general-purpose language***
- Subject-matter expert works here

- Reason about the model in a general-purpose way
- Computer scientists work here

Probabilistic Modeling Paradigm



Outline

1. Motivation: Where do probabilities come from, and why do we want to automatically reason about them?

2. How *probabilistic programs* make reasoning about probabilities easier
   - Modeling and querying

3. Challenges and research directions: why doesn't everyone use probabilistic programs today?
   - Scalability and language design

What makes a good modeling language?

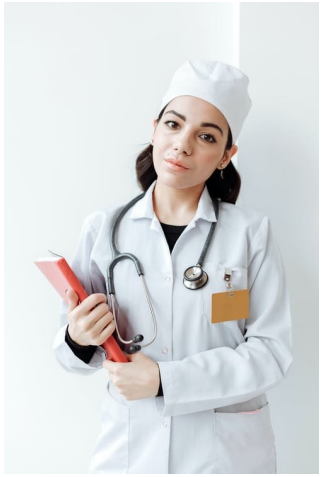
1. **Accessibility**: models should be easy to read and modify

2. **Modularity**: Languages should cleanly separate the modeling task from the querying task

3. **Expressivity**: Should be possible to expressed nuanced information about the world

Probabilistic Programming Languages

- ***Definition***: Probabilistic programs use the syntax and semantics of a *programming language* to define a probabilistic model



```
patient.flu ~ flip 0.001;
if(patient.flu) {
   patient.cough ~ flip(0.9);
   patient.temp ~ gaussian(101, 3);
} else {
…
```

- Goal: **Inference**, compute the probability that the program outputs a particular value

Your First Probabilistic Program

- It looks like this:

```
x ~ flip(0.5);
y ~ flip(0.7);
```

`x ~ flip(θ);` means "flip a coin, and output true with probability θ

- So we could "run" this program by evaluating each flip, then executing the program:

```
x ~ flip(0.5);
y ~ flip(0.7);
```

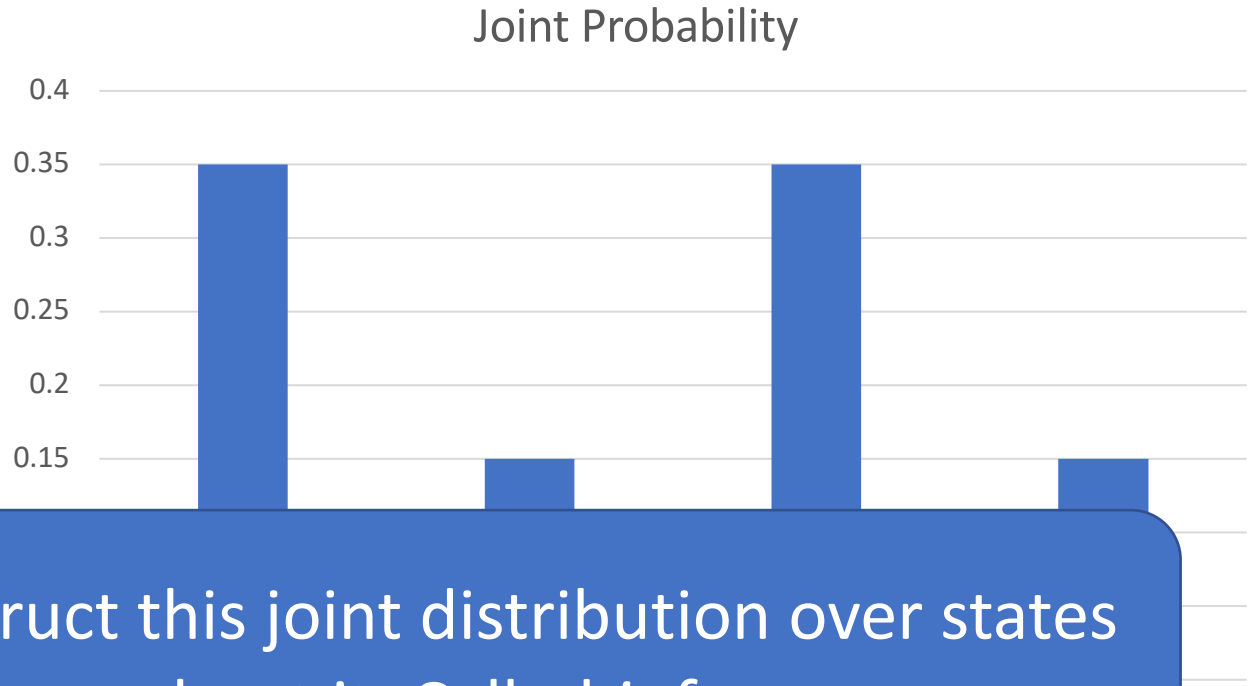
Sample

```
x = true;
y = false;
```

Probabilistic Program Semantics

- The program itself defines a probability distribution

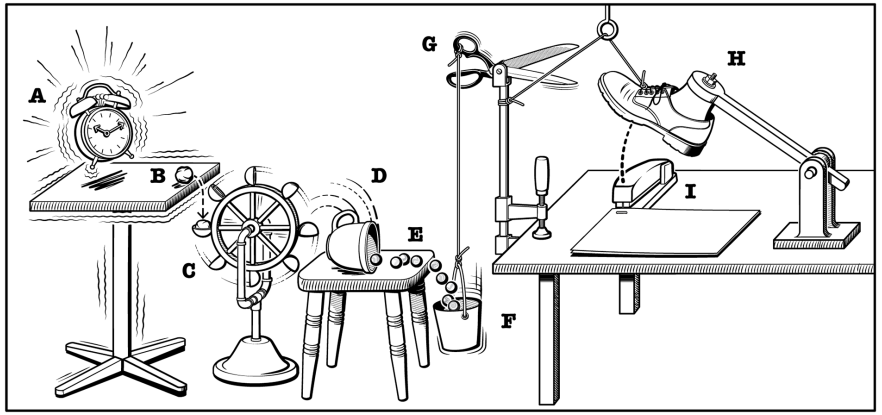
```
x ~ flip(0.5);
y ~ flip(0.7);
```



Goal: To construct this joint distribution over states and reason about it. Called *inference*.

The Power of Programming

```
if alarmRings {
    ballMoves ~ flip 0.99
} else {
    ballMoves ~ flip 0.01
}
ballLandsInHole ~ flip 0.99;
wheelTurns ~ flip 0.99;
if ballLandsInHole &&
wheelTurns {
    ...
}
...
```



- Describe the system and each of the components as random events
- The system handles reasoning automatically

PROGRAMMERS

Population: 100s of millions

People who use probabilistic models

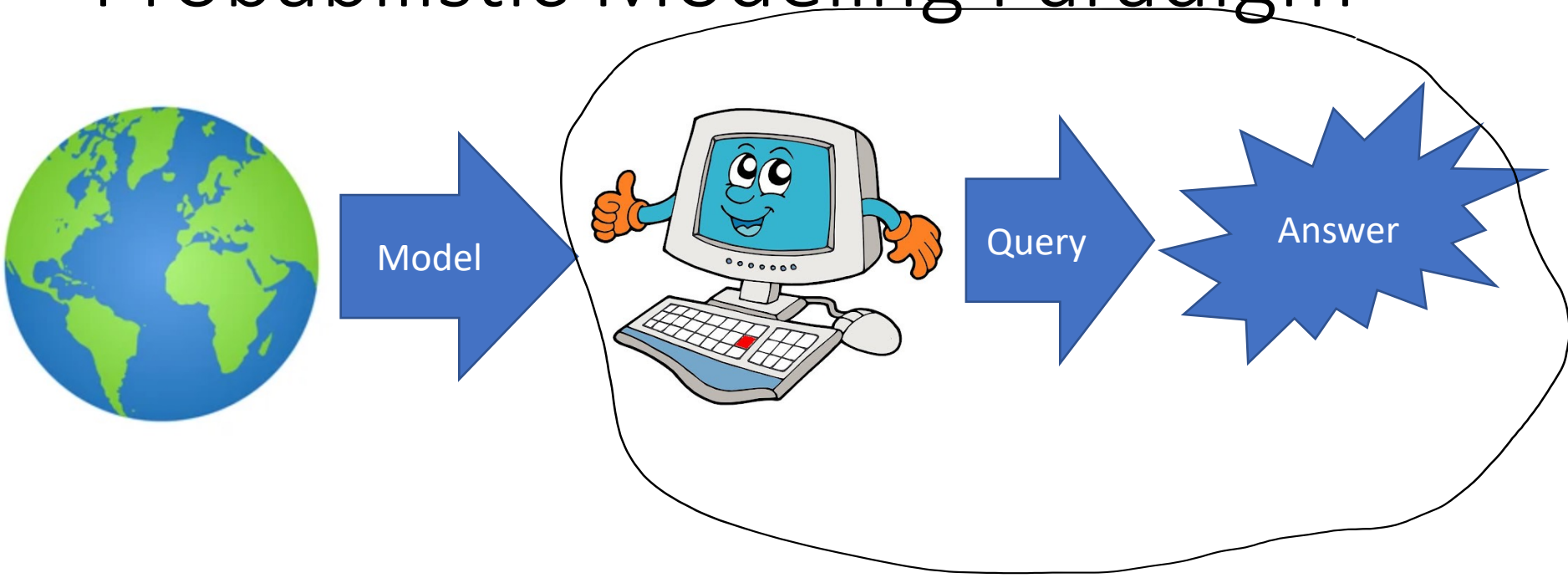
Google

Population: millions

Experts in probabilistic modeling

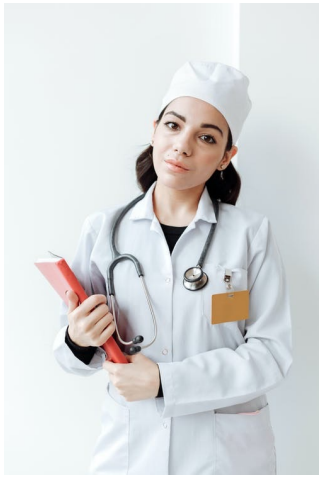
Population: thousands?

Probabilistic Modeling Paradigm



What kinds of queries are there, and what are they used for?

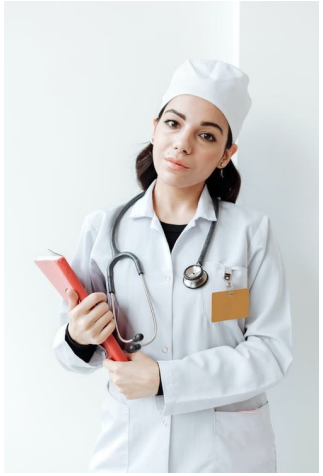
Kinds of Queries



```
patient.flu ~ flip 0.001;
Patient.cold ~ flip 0.005
if(patient.flu) {
   patient.cough ~ flip(0.9);
   patient.temp ~ gaussian(101, 3);
} else  if patient.cold {
    …
} else {
    …
}
```

- **Marginal Probability:** What is the probability that a patient has a cough?

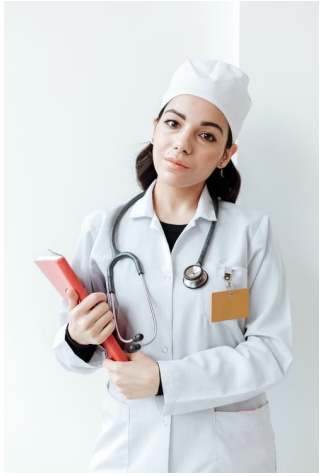
Kinds of Queries



```
patient.flu ~ flip 0.001;
Patient.cold ~ flip 0.005
if(patient.flu) {
   patient.cough ~ flip(0.9);
   patient.temp ~ gaussian(101, 3);
} else  if patient.cold {
    …
} else {
    …
}
```

- **Conditional Probability:** What is the probability that a patient has a cold given that they have a cough?

Kinds of Queries



```
patient.flu ~ flip 0.001;
Patient.cold ~ flip 0.005
if(patient.flu) {
   patient.cough ~ flip(0.9);
   patient.temp ~ gaussian(101, 3);
} else  if patient.cold {
    …
} else {
    …
}
```

- **Sensitivity/Robustness:** If I adjust how likely an average patient is to have a flu, how does that affect their probability of having a cough?

Probabilistic Queries

- Research Program: Collect as many kinds of *useful, general* queries as possible
  - Driven by practical use-cases of PPLs
- System *automatically* answers these queries for a given model
  - Can be validated and developed independently of the model by a separate team

Outline

1. Motivation: Where do probabilities come from, and why do we want to automatically reason about them?

2. How *probabilistic programs* make reasoning about probabilities easier
   - Some case studies of languages and their applications

3. Challenges and research directions: why doesn't everyone use probabilistic programs today?
   - Scalability and language design
   - What I do ☺

Today's PPL Development Lifecycle

Scientist (or other SME) has a problem

1

- Programmer attempts to implement their model in a PPL

2

- Programmer asks the system a query "What is the probability of X?"

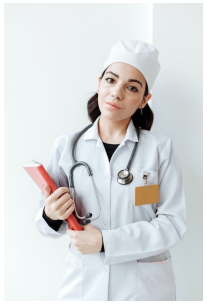
3

- Programmer gets an answer

Sometimes one of these steps fails, which guides language design

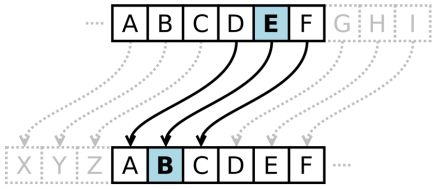
A Case Study in Language Development: Discreteness

1. Programs are naturally discrete (`if`-statements)



```
patient.flu = flip 0.001
if(patient.flu) {
…
} else {
…
```

2. Problems are naturally discrete



Challenge of Discreteness

Scientist (or other SME) has a problem

1

These steps work reasonably well for discrete programs

- Programmer attempts to implement their model in a PPL

2

- Programmer asks the system a query
  What is the probability of X?

3

- Programmer gets an answer

These steps often fail for most existing probabilistic programming languages!

Discreteness breaks many PPLs

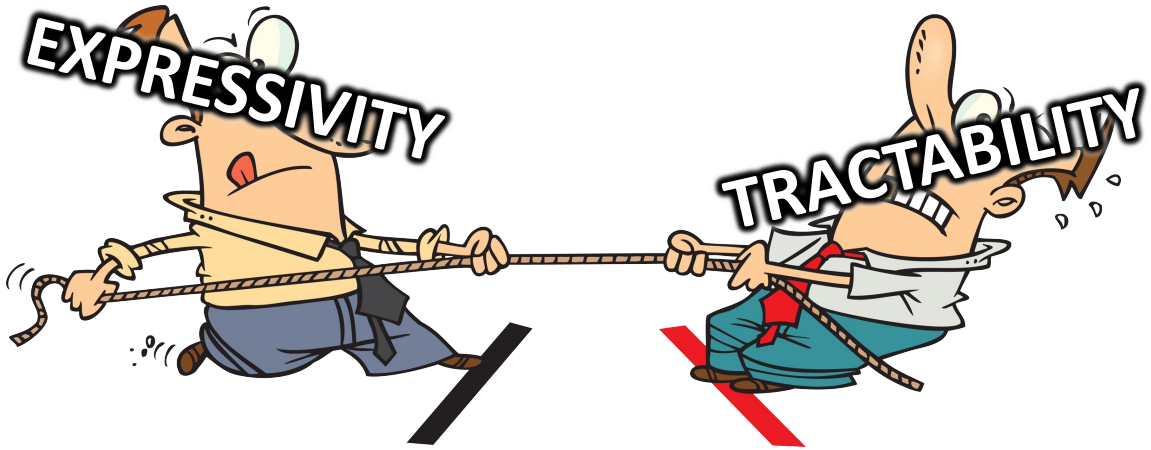
Does not support (general) `if`-statements!

coroutines. Whenever a discrete variable is encountered in a program's execution, the program is suspended and resumed multiple times with all possible values in the support of that distribution. Listing 10, which implements a simple finite

WebPPL

[AADB+'19]

Central Tension of PPL Design



PPL Development Lifecycle

- We want to model discreteness…
  - But, most existing languages today cannot handle this kind of program very well
  - Insufficiently general-purpose

- So, we developed a new PPL that can scale to extremely large discrete programs
  - Pick one kind of problem and ***solve it well***

Dice

The dice probabilistic programming language

About GitHub Paper

http://dicelang.cs.ucla.edu

`dice` is a probabilistic programming language focused on fast exact inference for discrete probabilistic programs. For more information on `dice`, see the about page or paper.

Below is an online `dice` code demo. To run the example code, press the "Run" button. This will print the posterior probability of the encryption key used by a Caesar cipher given the observed string "CCCC".

Note that `dice` is still under active development, so there is always a chance of bugs: please help us by reporting them on github!

Run

```
1  fun n1(init: bool) {
2     let l1succeed = flip 0.99 in
3     let l2succeed = flip 0.91 in
4     init && l1succeed && l2succeed
5  }
6  fun n2(init: bool) {
7     let routeChoice = flip 0.5 in
8     if routeChoice then
9        init && flip 0.88 && flip 0.93
10    else
11       init && flip 0.19 && flip 0.33
12 }
13 n2(n2(n2(n2(n2(n2(n2(n2(n2(n2(n2(n2(n2(n2(n2(n2(n2(n2(n2(n1(true))))))))))))))))))))
14
```

Joint Distribution

Value	Probability
true	3.01006520877e-08
false	0.999999969899

Time (s)

- Designed by my lab
- Specialized for *discrete probabilistic programs*
- Can scale to megabyte-sized programs
- Hundreds of thousands of random variables

Design of Dice

- How can we scale to such large programs?
  - We *specialize* to only discrete programs

Growing the Landscape of PPLs

Takeaways

- Custom solutions for probabilistic reasoning do not scale
- The *probabilistic modeling paradigm* separates modeling from reasoning
  - Design modeling languages that domain experts can use
  - Experts create general-purpose reasoning backends
- Key challenges
  - Scaling inference to realistic systems
  - Designing languages that balance tractability and expressivity

Takeaways

- We want to help with your problems!
  - Helps *guide the development of PPLs* towards problems and domains that matter

References

- [**H**ZGTZ IROS'16] Steven Holtzen*, Yibiao Zhao*, Tao Gao, Josh Tenenbaum, and Song-Chun Zhu. *Inferring Human Intent from Video by Sampling Hierarchical Plans*. In IEEE International Conference on Intelligent Robots and Systems (IROS), 2016.
- [HJVMSB CAV'21] 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. To appear in 33rd International Conference on Computer-Aided Verification (CAV), 2021.
- [**H**VM OOPSLA'20] Steven Holtzen, Guy Van den Broeck, and Todd Millstein. *Scaling Exact Inference for Discrete Probabilistic Programs*. In Proc. ACM Program. Lang. 4 (OOPSLA), 2020.
- [Z**H**V UAI'20] Honghua Zhang, Steven Holtzen, and Guy Van den Broeck. *On the Relationship Between Probabilistic Circuits and Determinantal Point Processes*. In Proceedings of the 36th Conference on Uncertainty in Artificial Intelligence (UAI), 2020.
- [**H**MV UAI'19] Steven Holtzen, Todd Millstein, and Guy Van den Broeck. *Generating and Sampling Orbits for Lifted Probabilistic Inference*. In Proceedings of the 35th Conference on Uncertainty in Artificial Intelligence (UAI), 2019. Oral Presentation (35/450).

* Is equal contribution.

References

- [**H**VM ICML'18] Steven Holtzen, Guy Van den Broeck, and Todd Millstein. *Sound Abstraction and Decomposition of Probabilistic Programs*. In Proceedings of the 35th International Conference on Machine Learning (ICML), 2018.
- [**H**MV UAI'17] Steven Holtzen, Todd Millstein, and Guy Van den Broeck. *Probabilistic Program Abstractions*. In Proceedings of the 33rd Conference on Uncertainty in Artificial Intelligence (UAI), 2017.
- [H**H**MVM ASPLOS'21] Yipeng Huang, Steven Holtzen, Todd Millstein, Guy Van den Broeck, and Margaret R. Martonosi. *Noisy Variational Quantum Algorithm Simulation via Knowledge Compilation for Repeated Inference*. In Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2021.
- [**H**JCMSV (In Preparation)] Steven Holtzen*, Sebastian Junges*, Marcell Vazquez-Chanlatte, Todd Millstein, Sanjit A. Seshia, Guy Van den Broeck. *Model Checking Finite-Horizon Markov Chains with Probabilistic Inference.* Under review.
- [SGTVV SIGCOMM'20] Steffen, S., Gehr, T., Tsankov, P., Vanbever, L., & Vechev, M. (2020, July). Probabilistic Verification of Network Configurations. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication* (pp. 750-764).

* Is equal contribution.

References

- [AADB+'19] Ai, Jessica, Nimar S. Arora, Ning Dong, Beliz Gokkaya, Thomas Jiang, Anitha Kubendran, Arun Kumar, Michael Tingley, and Narjes Torabi. "HackPPL: a universal probabilistic programming language." In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pp. 20-28. 2019.
- [DJKV CAV'17] Dehnert, C., Junges, S., Katoen, J. P., & Volk, M. (2017, July). A storm is coming: A modern probabilistic model checker. In *International Conference on Computer Aided Verification* (pp. 592-600). Springer, Cham.
- [CVV'21] Choi, YooJung, Antonio Vergari, and Guy Van den Broeck. "Probabilistic Circuits: A Unifying Framework for Tractable Probabilistic Models."
- [H90] T. Herman. Probabilistic self-stabilization. Information Processing Letters, 35(2):63–67, 1990. An extended version is available at ftp://ftp.math.uiowa.edu/pub/selfstab/H90.html.
- [FSC+ PloS one '18] Fried, E. S., Sawaya, N. P., Cao, Y., Kivlichan, I. D., Romero, J., & Aspuru-Guzik, A. (2018). qTorch: The quantum tensor contraction handler. *PloS one*, *13*(12), e0208510.

* Is equal contribution.

References

- [PRKO AISTATS‘18] Pfeffer, A., Ruttenberg, B., Kretschmer, W., & OConnor, A. (2018, March). Structured factored inference for probabilistic programming. In *International Conference on Artificial Intelligence and Statistics* (pp. 1224-1232). PMLR.
- [CVV ‘20] YooJung Choi, Antonio Vergari and Guy Van den Broeck. Probabilistic Circuits: A Unifying Framework for Tractable Probabilistic Models, 2020.