# Introduction to Probabilistic Programming

SWS SEMINAR, 30 AUGUST 2


DARIO STE

dario.stein@ru.nl

damast93


# About me

dario.stein@ru.nl

damast93

2022 – present: Postdoc at iHub (Bart Jacobs)

2017 – 2022: PhD Computer Science (Sam Staton, Univ of Oxford)

Before: Pure Maths (Hamburg, Cambridge)

Interests:

Programming language semantics, **probabilistic programming**

Category theory, **categorical probability theory**, logic & type theory, quantum computation

Looking into: Theorem Proving, ML

# What is Probabilistic Programming?

Probabilistic Programming Languages (PPL) are the next generation programming systems for **statistical inference** with first-class probabilistic primitives.

Two goals:

1. Write down **flexible generative statistical models** with ease (Modeling, Communication)
2. Solve them **automatically** (Inference)

# Recent Interest

| Language | Ecosystem |
|---|---|
| Stan | |
| BUGS/JAGS* | |
| **WebPPL** | JavaScript |
| LazyPPL | Haskell |
| MonadBayes | Haskell |
| Infer.NET (*Microsoft*) | C# |
| Pyro (*Uber*) | Python |
| PyMC3 | Python |
| Bean Machine (*Meta*) | Python |

| Language | Ecosystem |
|---|---|
| Edward 1-2 (*Google*) | Python |
| Gen | Julia |
| Turing.jl | Julia |
| Anglican/Church | Clojure/Scheme |
| ProbLog | Prolog |
| BLOG* | |
| Hakaru | |
| Birch | |
| … | |

Many more: @Wikipedia „Probabilistic Programming“

# The PPL Workflow: Three primitives

1. Write down a generative statistical model
2. Feed in observations
3. Learn from the observations
4. … repeat

$$\text{PPL} = \text{sample} + \underbrace{\text{observe} + \text{infer}}_{\text{Magic!?}}$$

any old language with
random number generation

**Goal:**

- Sample from (approximate) posterior distribution
- Compute expectations or probabilities

# Strengths of Probabilistic Programming

*observe* + *infer* are fully integrated into one language

- Models can be **any program**
- Nested inference  Reasoning about Reasoning

Models are expressive + highly flexible

- Fewer parameters, high explainability

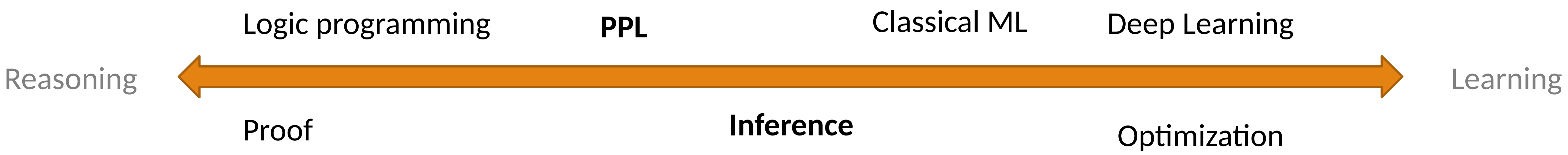
Clarity & ease of use

- Accessibility for domain experts
- Easy adaption and exploration of models

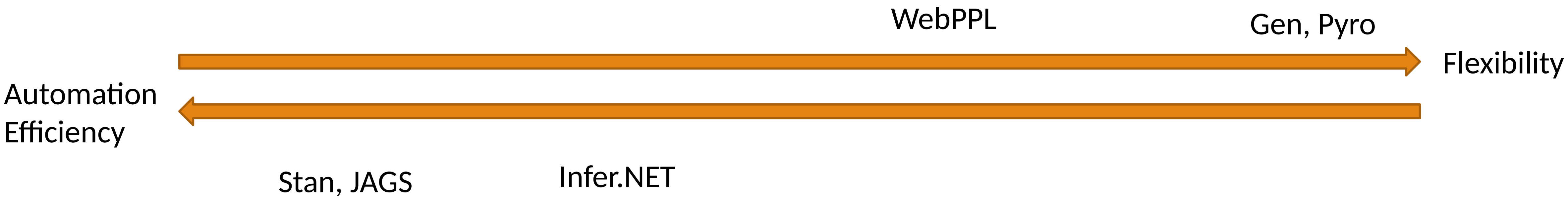
**Applications**

- Stats/ML: „Bayesian Machine Learning“
- Planning as Inference
- Epidemiology
- Neuroscience
- Linguistics/theory of mind/communication
- Program Synthesis
- Education?

# Learnings vs. Reasoning



# Tradeoff: Flexibility vs Efficiency



# Why WebPPL

http://webppl.org/

No setup, runs in your browser

Familiar language (JavaScript + observe + infer)

Out-of-the-box visualization

**Excellent resources**

- *Probabilistic Models of Cognition*: Free interactive book http://probmods.org/
- *The Design and Implementation of Probabilistic Programming Languages* http://dippl.org/

# Recap: Bayesian Inference

# Recap: Bayesian inference

Problem:

**Prior**: **10%** of population has covid

**Model**: Tests have **80%** sensitivity (true positive rate) and **98%** specificity (true negative)

**Evidence**: 1 positive test

**Bayesian inference**

**Posterior**: **82%** probability of covid

# Bayes‘ law

$$\underbrace{P(\text{hypothesis} \mid \text{evidence})}_{\text{posterior}} = \frac{\underbrace{P(\text{evidence} \mid \text{hypothesis})}_{\text{likelihood}}\ \underbrace{P(\text{hypothesis})}_{\text{prior}}}{P(\text{evidence})}$$

Thomas Bayes

# Bayes' law

$$P(covid \mid pos) = \frac{P(pos \mid covid)\, P(covid)}{P(pos)} = \frac{P(pos \mid covid)\, P(covid)}{P(pos \mid covid)\, P(covid) + P(pos \mid \neg covid) P(\neg covid)}$$

$$= \frac{0.8 * 0.1}{0.8 * 0.1 + 0.02 * 0.9} = 0.816$$

10% (Prior) $\xrightarrow{\text{Bayesian inference}}$ 82% (Posterior)

# Changing the model

Let‘s make the following changes to the model

- What about 1 positive + 1 negative result?
- 2 positive + 1 negative?
- … multiple symptoms
- … multiple competing diseases

ㅍ don‘t do this by hand!

# Inference with WebPPL

# Inference with WebPPL

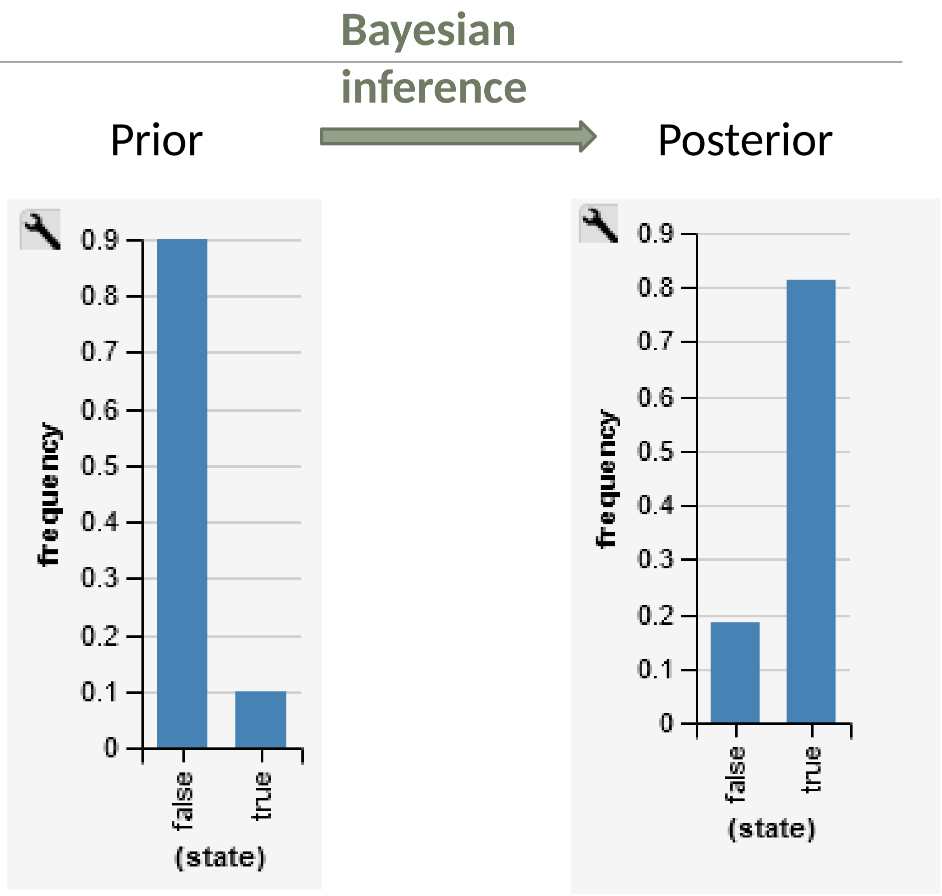
Covid test result as a stochastic function of the underlying condition

```
var test = function(has_covid) {
  var positive_prob = has_covid
    ? 0.8
    : 0.02

  return flip(positive_prob)
    ? 'pos’
    : 'neg'
}
```

```
// Model
var has_covid = flip(0.1)
var test_result = test(has_covid)

// Observation
condition(test_result == 'pos')

// Prediction
return has_covid
```

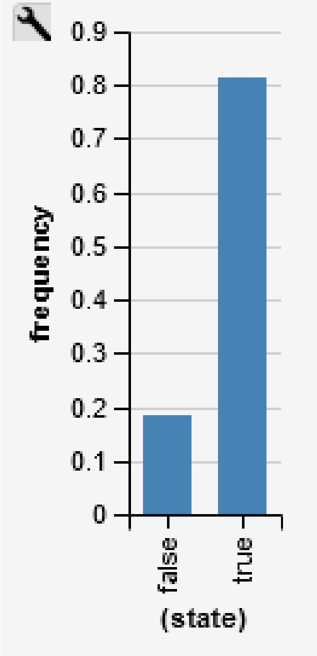
# Full Program

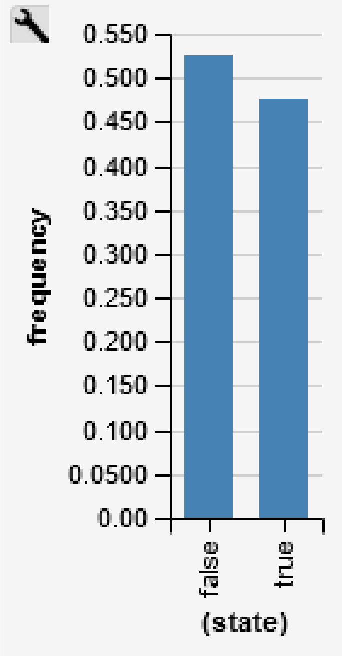
```
var test = function(has_covid) {
  var positive_prob = has_covid ? 0.8 : 0.02
  return flip(positive_prob) ? 'pos' : 'neg'
}

viz(Infer(function() {
    // Model
    var has_covid = flip(0.1)
    var test_result = test(has_covid)

    // Observation
    condition(test_result == 'pos')

    // Prediction
    return has_covid
  }
))
```

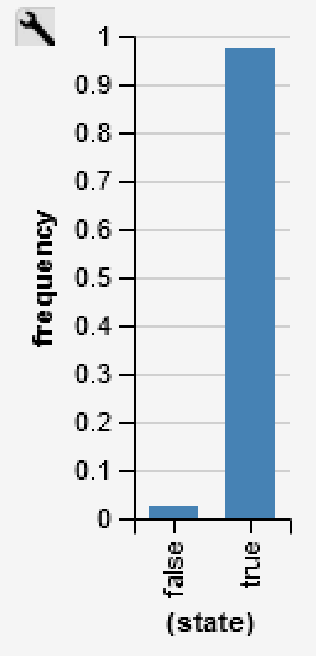


# Easy to add more observations

```
var test = function(has_covid) {
  var positive_prob = has_covid ? 0.8 : 0.02
  return flip(positive_prob) ? 'pos' : 'neg'
}

viz(Infer(function() {
    // Model
    var has_covid = flip(0.1)

    // Observation
    condition(test(has_covid) == 'pos')
    condition(test(has_covid) == 'neg')
    condition(test(has_covid) == 'pos’)

    // Prediction
    return has_covid
  }
))
```



# Implementation

# Inference algorithms

Inference is a hard problem (strictly harder than optimization).

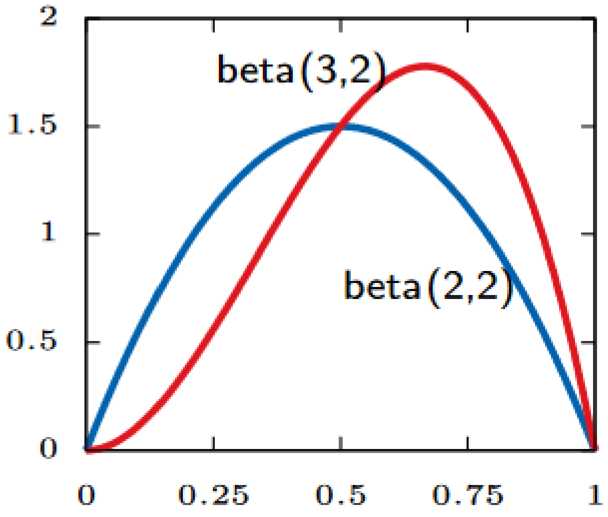
Different problems require **different types of algorithms**

But inference methods can easily be **interchanged**

```
Infer({method: “enumerate”, …}, function() {
```

# Example with continuous variables

```
viz(Infer({method: 'rejection'}, function() {
  var p = beta({a:2, b:2})
  condition(flip(p) == true)

  return p
}))
```



# Inference algorithms

Inference is a hard problem (strictly harder than optimization).

But inference methods can easily be swapped

```
Infer({method: “enumerate”, …}, function() {
```

Exact inference (exhaustive enumeration, symbolic inference)

Optimization (Variational inference, EM)

Simulation (Monte-Carlo Methods)

- Rejection sampling
- Likelihood-weighted importance sampling
- Markov-Chain Monte Carlo (Metropolis-Hastings, HMC)
- Particle Filters

# Implementation

Need for composable abstractions

- *sample* and *observe* are purely **abstract primitives**, and have different meanings depending on the inference algorithm. E.g.
  - Importance sampling → Generate random trace, record a likelihood factor
  - Enumeration → call with all possible values
  - MCMC → sample creates a resumable entry point
- Easiest to build on top of a purely functional language
- CPS transform, or even better: monads/effect handlers

# Implementation

e.g. `Control.Monad.Bayes` or [Ścibior'2017]

```
class (Monad m) => MonadInfer m where
    flip :: Double -> m Bool
    score :: Double -> m ()

data Dist a = Dist [(a,Double)] – for enumeration

instance MonadInfer Dist where
   flip p  = Dist [(True,p), (False,1-p)]
   score r = Dist [( (), r)]
```

# Implementation

Need for composable abstractions

- *sample* and *observe* are purely **abstract primitives**, and have different meanings depending on the inference algorithm. E.g.
  - Importance sampling ▭ Generate random trace, record a likelihood factor
  - Enumeration ▭ call with all possible values
  - MCMC ▭ sample creates a resumable entry point
- Easiest to build on top of a purely functional language
- CPS transform, or even better: monads/effect handlers

Programmable inference (Gen/Pyro)

- modular building blocks for customizable inference algorithms
- *guide programs* for variational inference

# Compositionality

Implicature in Linguistics: Saying „some“ probably doesn‘t mean „all“

We model this using **nested inference**

```
var speaker = function(state, depth) {
  return Infer({method: 'enumerate'},
function() {
    var words = sample(sentencePrior)
    condition(state == sample(listener(words,
depth)))
    return words
  })
};
```

```
var listener = function(words, depth) {
  return Infer({method: 'enumerate'}, function() {
    var state = sample(statePrior);
    var wordsMeaning = meaning(words)
    condition(depth == 0 ? wordsMeaning(state) :
                _.isEqual(words,
sample(speaker(state, depth - 1))))
    return state;
  })
};
```

http://probmods.org/chapters/social-cognition.html

# Verification

Verifying implementations of PPLs is tricky: E.g. creative use of laziness (LazyPPL), autodiff etc.

Program transformation to increase efficiency

```
var p = beta({a:1, b:1})

// rejects frequently
observe(flip(p) == true)
```

```
var p = beta({a:1, b:1})

// score by likelihood
factor(p)
```

```
var p = beta({a:2, b:1}
```

Denotational semantics ㄹㅌ Categorical probability theory

Foundational requirements diverge from usual mathematical probability (measure theory):

- **Quasi-Borel spaces** [Heunen&al'17] for random higher-order functions

# Summary

Statistics literature is notoriously difficult to read

When discussing statistical questions – do it in code!

- Unified langugage for modeling and observations
- Encode your domain knowledge (law, humanities, science)
- Quickly run, adapt, explore sophisticated models

Everyone can use PPL as a way to explore and communicate statistics. Try it in your browser!

- [http://webppl.org/](http://webppl.org/)
- [http://probmods.org](http://probmods.org)

# Interactive Examples

# Interactive Examples

Real-world example due to Hanna Schraffenberger

Legal Example

**Advanced [from probmods.org]**

Category Learning: http://probmods.org/chapters/hierarchical-models.html

Learning Logical Explanations (Occam‘s Razor): http://probmods.org/chapters/lot-learning.html

Reasoning about Reasoning, Linguistic Implicature: http://probmods.org/chapters/social-cognition.html

# Hanna‘s Problem

“We had an experiment with 1000 participants, divided over three general conditions A, B and C. After the experiment we ask them an attention-check question to see if they paid attention and remember their condition correctly ("did you see A, B or C"?). It now turns out that:

- 200 people give an incorrect answer to this question

- 400 said they cannot remember

- 400 had it correct”

**Question: How many people just guessed?**

# A Legal Example

Police arrest suspects A,B,C,D,E.

- Testimony: „I am 80% sure the perpetrator is among A,B,C,D.“
- A,B,C are found to have alibis

Problem: What‘s the probability that D is the perpetrator?

**Quiz**:

(I) 80%

(II) 50%

# A Legal Example

Police arrest suspects A,B,C,D,E.

- Testimony: „I am 80% sure the perpetrator is among A,B,C,D.“
- A,B,C are found to have alibis

Problem: What‘s the probability that D is the perpetrator?

How to formally interpret such statements?

Why does the testimony even give new information? (Same if you guess!)