Paradoxes of Probabilistic Programming (POPL'21)

and deleted scenes (VeriProP'21)

Jules Jacobs

Radboud University Nijmegen
julesjacobs@gmail.com

These slides: julesjacobs.com/slides/veriprop2021.pdf

Probabilistic programming

Example:

- A scientist randomly selects a man and a woman and measures their height
- The woman's height $h \sim \mathit{Normal}(1.7, 0.5)$ meters
- The man's height $h' \sim \mathit{Normal}(1.8, 0.5)$ meters

Question: What's the expectation of h conditioned on $h' = h$?

Probabilistic programming

Example:

- A scientist randomly selects a man and a woman and measures their height
- The woman's height $h \sim \mathit{Normal}(1.7, 0.5)$ meters
- The man's height $h' \sim \mathit{Normal}(1.8, 0.5)$ meters

Question: What's the expectation of h conditioned on $h' = h$?

```
function meters(){
  h = rand(Normal(1.7, 0.5))
  observe(Normal(1.8, 0.5), h)
  return h
}
samples = run(meters, 1000)
estimate = average(samples)
```

Answer: ≈ 1.75

Probabilistic programming

Example:

- A scientist randomly selects a man and a woman and measures their height
- The woman's height $h \sim \mathit{Normal}(1.7, 0.5)$ meters
- The man's height $h' \sim \mathit{Normal}(1.8, 0.5)$ meters

Question: What's the expectation of h conditioned on $h' = h$?

```
function meters(){
  h = rand(Normal(1.7, 0.5))
  observe(Normal(1.8, 0.5), h)
  return h
}
samples = run(meters, 1000)
estimate = average(samples)
```

Answer: ≈ 1.75

```
function centimeters(){
  h = rand(Normal(170, 50))
  observe(Normal(180, 50), h)
  return h
}
samples = run(centimeters, 1000)
estimate = average(samples)
```

Answer: ≈ 175

Paradox

```
h = rand(Normal(1.7, 0.5))
w = rand(Normal(60, 10))
if(flip(0.5)){
  observe(Normal(1.8, 0.5), h)
}else{
  observe(Normal(70, 10), w)
}
return h
```

Answer: ≈ 1.75

Paradox

```
h = rand(Normal(1.7, 0.5))
w = rand(Normal(60, 10))
if(flip(0.5)){
  observe(Normal(1.8, 0.5), h)
}else{
  observe(Normal(70, 10), w)
}
return h
```

Answer: ≈ 1.75

```
h = rand(Normal(170, 50))
w = rand(Normal(60, 10))
if(flip(0.5)){
  observe(Normal(180, 50), h)
}else{
  observe(Normal(70, 10), w)
}
return h
```

Answer: ≈ 170

Paradox

```
h = rand(Normal(1.7, 0.5))
w = rand(Normal(60, 10))
if(flip(0.5)){
  observe(Normal(1.8, 0.5), h)
}else{
  observe(Normal(70, 10), w)
}
return h
```

Answer: ≈ 1.75

```
h = rand(Normal(170, 50))
w = rand(Normal(60, 10))
if(flip(0.5)){
  observe(Normal(180, 50), h)
}else{
  observe(Normal(70, 10), w)
}
return h
```

Answer: ≈ 170

- The output depends on whether we use meters or centimeters
- Happens in implementations as well as in formal operational semantics
- Similar behaviour in programs without conditionals too (Borel-Komolgorov paradox)

Problem:

- Probabilistic programs are not invariant under parameter transformations
- It's not clear what `observe` really means

Problem:

- Probabilistic programs are not invariant under parameter transformations
- It's not clear what `observe` really means

Key ideas:

1. Determine what `observe` *should* mean by looking at positive measure conditioning
2. Change the language: `observe` conditions on *intervals* instead of points:
   `observe(Normal(1.8, 0.5), Interval(h, 0.1))`
3. Parameterize the program by `eps`:
   ```
   function foo(eps){
     ... observe(Normal(1.8, 0.5), Interval(h, eps)) ...
   }
   ```
4. Take the limit `eps` $\to 0$.
5. Use symbolic infinitesimal arithmetic to compute the limit.

Problem:

- Probabilistic programs are not invariant under parameter transformations
- It's not clear what `observe` really means

Key ideas:

1. Determine what `observe` *should* mean by looking at positive measure conditioning
2. Change the language: `observe` conditions on *intervals* instead of points:
   `observe(Normal(1.8, 0.5), Interval(h, 0.1))`
3. Parameterize the program by `eps`:
   ```
   function foo(eps){
       ... observe(Normal(1.8, 0.5), Interval(h, eps)) ...
   }
   ```
4. Take the limit `eps` $\to 0$.
5. Use symbolic infinitesimal arithmetic to compute the limit.

Result:

- New language is invariant under arbitrary parameter transformations
- Programs have clear probabilistic meaning via *rejection sampling*
- Implemented as a DSL in Julia

Paradox revisited

```
A = 2.3 // meters          B = 42.6 // kilograms
function foo(eps){
  h = rand(Normal(1.7, 0.5))  // meters
  w = rand(Normal(60, 10))  // kilograms
  if(flip(0.5)){
    observe(Normal(1.8, 0.5), Interval(h,A*eps))
  }else{
    observe(Normal(70, 10), Interval(w,B*eps))
  }
  return h
}
```

Paradox revisited

```
A = 2.3 // meters          B = 42.6 // kilograms
function foo(eps){
  h = rand(Normal(1.7, 0.5))  // meters
  w = rand(Normal(60, 10))  // kilograms
  if(flip(0.5)){
    observe(Normal(1.8, 0.5), Interval(h,A*eps))
  }else{
    observe(Normal(70, 10), Interval(w,B*eps))
  }
  return h
}
```

- Assume *rejection sampling* as "gold standard" semantics (works because width > 0) `observe(D,Interval(x,w))` $\triangleq$ reject if `random(D)` $\notin [x - w, x + w]$
- Try `foo(0.1)`; `foo(0.01)`; `foo(0.001)` for different values of `A` and `B`
- The relative size of `A` and `B` matters, even as `eps` $\to 0$
- Change of units of `h` and `w` requires corresponding change in interval width `A` and `B`

Non-linear parameter transformations

- The problem is more general than units and conditionals
- The general issue is invariance under parameter transformations
- Changes of units = linear parameter transformations
  $\implies$ produces paradoxes in combination with conditionals
- General case: non-linear parameter transformations (e.g. log-transform)
  $\implies$ produces paradoxes even without conditionals (e.g. Borel-Komolgorov paradox)
- See paper "Paradoxes of probabilistic programming" for details (https://julesjacobs.com/pdf/paradoxes.pdf)

Implementation

- Semantics: *rejection sampling*
  `observe(D,I)` $\triangleq$ reject if `random(D)` $\notin$ `I`
- Implementation: *likelihood weighting*
  `observe(D,I)` $\triangleq$ `{ weight *= P(D,I) }` where `P(D,I)` $\triangleq \mathbb{P}($ `random(D)` $\in$ `I` $)$.
- The interval `I` can depend on `eps`
  $\implies$ to compute lim `eps` $\to 0$ exactly, do arithmetic with $\mathbb{R}_\epsilon \triangleq \{a\epsilon^n \mid a \in \mathbb{R}, n \in \mathbb{Z}\}$
- Similar to automatic differentiation with dual numbers
- Dual numbers: $a + b\epsilon$ where $\epsilon^2 = 0$
- Infinitesimal probabilities: $a\epsilon^n$ where $1 + \epsilon = 1$

Implementation

- Semantics: *rejection sampling*

  `observe(D,I)` $\triangleq$ reject if `random(D)` $\notin$ `I`
- Implementation: *likelihood weighting*

  `observe(D,I)` $\triangleq$ `{ weight *= P(D,I) }` where `P(D,I)` $\triangleq \mathbb{P}($ `random(D)` $\in$ `I` $)$.
- The interval `I` can depend on `eps`

  $\implies$ to compute lim `eps` $\to 0$ exactly, do arithmetic with $\mathbb{R}_\epsilon \triangleq \{a\epsilon^n \mid a \in \mathbb{R}, n \in \mathbb{Z}\}$
- Similar to automatic differentiation with dual numbers
- Dual numbers: $a + b\epsilon$ where $\epsilon^2 = 0$
- Infinitesimal probabilities: $a\epsilon^n$ where $1 + \epsilon = 1$

Result:

- This `observe` is invariant under arbitrary parameter transformations:

  `observe(f(D), f(I))` $\equiv$ `observe(D, I)`
- Programs have clear probabilistic meaning via *rejection sampling*
- Can still condition on measure zero events
- Implemented as a DSL in Julia

Originally in the paper: Beyond intervals

We can let `I` in `observe(D,I)` be an arbitrary set
as long as we can compute `P(D,I)` $\triangleq \mathbb{P}($ `random(D)` $\in$ `I` $)$
e.g.

Originally in the paper: Beyond intervals

We can let `I` in `observe(D,I)` be an arbitrary set
as long as we can compute `P(D,I)` $\triangleq \mathbb{P}($ `random(D)` $\in$ `I` $)$
e.g.

- Union of intervals
- Finite set (if `D` is discrete)
- Regular language (if `D` is a Markov chain)
- General $I \subseteq \mathbb{R}^n$ for which we can approximate `P(D,I)` (if `D` multivariate)
  - e.g. ellipsoid $I_\epsilon \triangleq \{|A\vec{x} + b| \leq \epsilon \mid \vec{x} \in \mathbb{R}^n\}$
  - We can compute $P(D, I_\epsilon)$ for infinitesimal ϵ in terms of the PDF of D
  - For finite $\epsilon > 0$ we may need numerical integration

Originally in the paper: Soft observations

Generalize further: use soft indicator function $f : \Omega \to [0,1]$ instead of hard sets

- $f(x) \triangleq$ probability of accepting x
- Semantics: `observe(D,f)` $\triangleq$ reject if `flip(f(random(D))) == false`

Originally in the paper: Soft observations

Generalize further: use soft indicator function $f : \Omega \to [0, 1]$ instead of hard sets

- $f(x) \triangleq$ probability of accepting x
- Semantics: `observe(D,f)` $\triangleq$ reject if `flip(f(random(D))) == false`
- Implementation: `observe(D,f)` $\triangleq$ `{ weight *= W(D,f) }`
  where `W(D,f)` $\triangleq \int f(x) d\mathbb{P}(D)$
- e.g if f is piecewise constant, and we have a CDF for D, then we can compute $W(D, f)$
  - Such f specifies the rejection probability for each piecewise constant region

Originally in the paper: Soft observations

Generalize further: use soft indicator function $f : \Omega \to [0, 1]$ instead of hard sets

- $f(x) \triangleq$ probability of accepting x
- Semantics: `observe(D,f)` $\triangleq$ reject if `flip(f(random(D))) == false`
- Implementation: `observe(D,f)` $\triangleq$ `{ weight *= W(D,f) }`
  where `W(D,f)` $\triangleq \int f(x) d\mathbb{P}(D)$
- e.g if f is piecewise constant, and we have a CDF for D, then we can compute $W(D, f)$
  - Such f specifies the rejection probability for each piecewise constant region
- Note: f is *not* a probability density function.
  - Probability density functions *integrate* to 1
  - Soft observation functions return a *probability* (possibly infinitesimal)
  - The PDF of the normal distribution is not a soft indicator function, but $\sin(x)^2$ is

Originally in the paper: Events

Generalize further: use `observe(D)` where `D` $=$ Bernoulli(p)

- Semantics: `observe(Bernoulli(p))` $\triangleq$ reject if `flip(p) == false`
- Implementation: `observe(Bernoulli(p))` $\triangleq$ `{ weight *= p }`
- We view Bernoulli(p) as a "random boolean" and we observe that the boolean is true
- Probability p is allowed to be infinitesimal

Originally in the paper: Events

Generalize further: use `observe(D)` where `D` $=$ Bernoulli(p)

- Semantics: `observe(Bernoulli(p))` $\triangleq$ reject if `flip(p) == false`
- Implementation: `observe(Bernoulli(p))` $\triangleq$ `{ weight *= p }`
- We view Bernoulli(p) as a "random boolean" and we observe that the boolean is true
- Probability p is allowed to be infinitesimal
- Define `within(D,I)` $\triangleq$ Bernoulli($\mathbb{P}($ `random(D)` $\in$ `I` $)$)
  - Recovers `observe(D,I)` as `observe(within(D,I))`

Originally in the paper: Events

Generalize further: use `observe(D)` where `D` $=$ Bernoulli(p)

- Semantics: `observe(Bernoulli(p))` $\triangleq$ reject if `flip(p) == false`
- Implementation: `observe(Bernoulli(p))` $\triangleq$ `{ weight *= p }`
- We view Bernoulli(p) as a "random boolean" and we observe that the boolean is true
- Probability p is allowed to be infinitesimal
- Define `within(D,I)` $\triangleq$ Bernoulli($\mathbb{P}($ `random(D)` $\in$ `I` $)$)
  - Recovers `observe(D,I)` as `observe(within(D,I))`
- Boolean operations on Bernoulli's:

```
E1 = within(D1,I1)
E2 = within(D2,I2)
observe(or(E1,not(E2)))
```

- Rejection sampling semantics:

```
if(!(random(D1) in I1 || random(D2) notin I2)){ reject(); }
```

Comments or questions?

julesjacobs@gmail.com

These slides: julesjacobs.com/slides/veriprop2021.pdf

Acknowledgements I thank Sriram Sankaranarayanan and the anonymous POPL reviewers for their outstanding feedback. I'm grateful to Ike Mulder, Arjen Rouvoet, Paolo Giarrusso, Dongho Lee, Ahmad Salim Al-Sibahi, Sam Staton, Christian Weilbach, Johannes Borgström, Alex Lew, and Robbert Krebbers for help, inspiration, and discussions.