Paradoxes of Probabilistic Programming

and how to condition on events of measure zero with infinitesimal probabilities (POPL'21)

Jules Jacobs

Radboud University Nijmegen
mail@julesjacobs.com

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 1

Suppose the scientist is lazy, and only does the measurement half of the time...

Paradox 1

Suppose the scientist is lazy, and only does the measurement half of the time...

Meters:

```
h = rand(Normal(1.7, 0.5))
if(flip(0.5)){
  observe(Normal(1.8, 0.5), h)
}
return h
```

Answer: ≈ 1.721

Paradox 1

Suppose the scientist is lazy, and only does the measurement half of the time...

Meters:

```
h = rand(Normal(1.7, 0.5))
if(flip(0.5)){
  observe(Normal(1.8, 0.5), h)
}
return h
```

Answer: ≈ 1.721

Centimeters:

```
h = rand(Normal(170, 50))
if(flip(0.5)){
  observe(Normal(180, 50), h)
}
return h
```

Answer: ≈ 170.2

- The answer depends on whether the scientist uses meters or centimeters!

Paradox 1

Suppose the scientist is lazy, and only does the measurement half of the time...

Meters:

```
h = rand(Normal(1.7, 0.5))
if(flip(0.5)){
  observe(Normal(1.8, 0.5), h)
}
return h
```

Answer: ≈ 1.721

Centimeters:

```
h = rand(Normal(170, 50))
if(flip(0.5)){
  observe(Normal(180, 50), h)
}
return h
```

Answer: ≈ 170.2

- The answer depends on whether the scientist uses meters or centimeters!
- Happens if we run this with importance sampling

Paradox 1

Suppose the scientist is lazy, and only does the measurement half of the time...

Meters:

```
h = rand(Normal(1.7, 0.5))
if(flip(0.5)){
  observe(Normal(1.8, 0.5), h)
}
return h
```

Answer: ≈ 1.721

Centimeters:

```
h = rand(Normal(170, 50))
if(flip(0.5)){
  observe(Normal(180, 50), h)
}
return h
```

Answer: ≈ 170.2

- The answer depends on whether the scientist uses meters or centimeters!
- Happens if we run this with importance sampling
- Even happens in formal operational semantics (e.g. Commutative or Quasi-Borel)

Paradox 1

Suppose the scientist is lazy, and only does the measurement half of the time...

Meters:

```
h = rand(Normal(1.7, 0.5))
if(flip(0.5)){
  observe(Normal(1.8, 0.5), h)
}
return h
```

Answer: ≈ 1.721

Centimeters:

```
h = rand(Normal(170, 50))
if(flip(0.5)){
  observe(Normal(180, 50), h)
}
return h
```

Answer: ≈ 170.2

- The answer depends on whether the scientist uses meters or centimeters!
- Happens if we run this with importance sampling
- Even happens in formal operational semantics (e.g. Commutative or Quasi-Borel)
- Unclear what the answer *should* be, or whether this program should be disallowed

Paradox 2

Objection: you shouldn't do observe a variable number of times based on coin flip

Suppose the scientist is drunk, and measures the weight half of the time...

Paradox 2

Objection: you shouldn't do observe a variable number of times based on coin flip

Suppose the scientist is drunk, and measures the weight half of the time...

```
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 2

Objection: you shouldn't do observe a variable number of times based on coin flip

Suppose the scientist is drunk, and measures the weight half of the time...

```
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 same number of observes regardless of the outcome of the coin flip
- The output still depends on whether we use meters or centimeters

Paradox 3 (similar to Borel-Komolgorov)

Objection: you shouldn't do observe inside a conditional

Suppose the scientist uses a ruler marked in log scale...

Paradox 3 (similar to Borel-Komolgorov)

Objection: you shouldn't do observe inside a conditional

Suppose the scientist uses a ruler marked in log scale...

Original program:

```
h = rand(Normal(1.7,0.5))
observe(Normal(1.8,0.5),h)
return h
```

Answer: 1.75

Paradox 3 (similar to Borel-Komolgorov)

Objection: you shouldn't do observe inside a conditional

Suppose the scientist uses a ruler marked in log scale...

Original program:

```
h = rand(Normal(1.7,0.5))
observe(Normal(1.8,0.5),h)
return h
```

Answer: 1.75

Logarithmic ruler program:

```
H = rand(LogNormal(1.7,0.5))
observe(LogNormal(1.8,0.5),H)
return log(H)
```

Answer: 1.62

Paradox 3 (similar to Borel-Komolgorov)

Objection: you shouldn't do observe inside a conditional

Suppose the scientist uses a ruler marked in log scale...

Original program:

```
h = rand(Normal(1.7,0.5))
observe(Normal(1.8,0.5),h)
return h
```

Answer: 1.75

Logarithmic ruler program:

```
H = rand(LogNormal(1.7,0.5))
observe(LogNormal(1.8,0.5),H)
return log(H)
```

Answer: 1.62

- Whether we use linear scale or log scale shouldn't matter
- What do probabilistic programs really mean?

Overview

Problem:

- Output of probabilistic programs depends on the scale
- It's not clear what `observe` is supposed to mean

Overview

Problem:

- Output of probabilistic programs depends on the scale
- It's not clear what `observe` is supposed to mean

Key ideas:

1. Figure out what `observe` should do, by analogy with the discrete case
2. `observe` on *intervals* instead of points
3. Can condition on infinitesimally small intervals

Overview

Problem:

- Output of probabilistic programs depends on the scale
- It's not clear what `observe` is supposed to mean

Key ideas:

1. Figure out what `observe` should do, by analogy with the discrete case
2. `observe` on *intervals* instead of points
3. Can condition on infinitesimally small intervals

Result:

- No unit/scale anomalies
- Programs have clear probabilistic meaning via *rejection sampling*
- Proof of concept in Julia

Probabilistic programming 101: Rejection sampling

```
function threeDice(){
  x = rand(DiscreteUniform(1,6))
  y = rand(DiscreteUniform(1,6))
  z = rand(DiscreteUniform(1,6))
  observe(z == x + y)
  return x
}
samples = run(threeDice, 1000)
```

Probabilistic programming 101: Rejection sampling

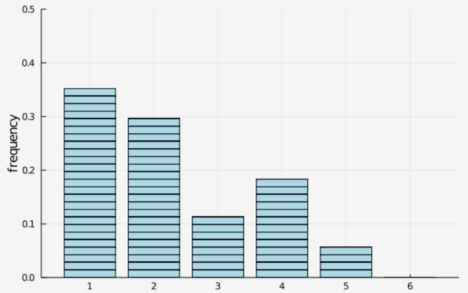
```
function threeDice(){
  x = rand(DiscreteUniform(1,6))
  y = rand(DiscreteUniform(1,6))
  z = rand(DiscreteUniform(1,6))
  observe(z == x + y)
  return x
}
samples = run(threeDice, 1000)
```

PPL implementation:

```
weight = 1
function observe(b){
  if(!b) weight = 0
}
function run(func, k){
  samples = []
  for(i in 1..k){
    weight = 1
    y = func()
    if(weight == 1){
      samples.add(y)
    }
  }
  return samples
}
```

Probabilistic programming 101: Rejection sampling

```
function threeDice(){
  x = rand(DiscreteUniform(1,6))
  y = rand(DiscreteUniform(1,6))
  z = rand(DiscreteUniform(1,6))
  observe(z == x + y)
  return x
}
samples = run(threeDice, 1000)
```

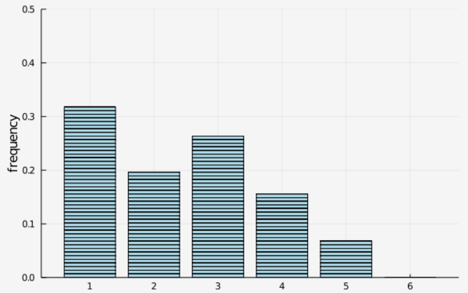


PPL implementation:

```
weight = 1
function observe(b){
  if(!b) weight = 0
}
function run(func, k){
  samples = []
  for(i in 1..k){
    weight = 1
    y = func()
    if(weight == 1){
      samples.add(y)
    }
  }
  return samples
}
```

Probabilistic programming 101: Rejection sampling

```
function threeDice(){
  x = rand(DiscreteUniform(1,6))
  y = rand(DiscreteUniform(1,6))
  z = rand(DiscreteUniform(1,6))
  observe(z == x + y)
  return x
}
samples = run(threeDice, 1000)
```

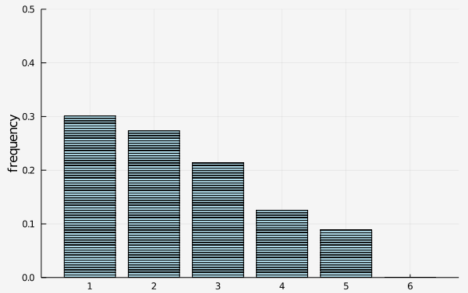


PPL implementation:

```
weight = 1
function observe(b){
  if(!b) weight = 0
}
function run(func, k){
  samples = []
  for(i in 1..k){
    weight = 1
    y = func()
    if(weight == 1){
      samples.add(y)
    }
  }
  return samples
}
```

Probabilistic programming 101: Rejection sampling

```
function threeDice(){
  x = rand(DiscreteUniform(1,6))
  y = rand(DiscreteUniform(1,6))
  z = rand(DiscreteUniform(1,6))
  observe(z == x + y)
  return x
}
samples = run(threeDice, 1000)
```

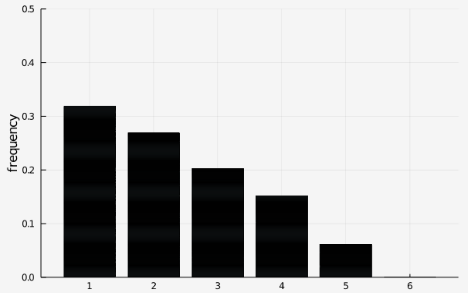


PPL implementation:

```
weight = 1
function observe(b){
  if(!b) weight = 0
}
function run(func, k){
  samples = []
  for(i in 1..k){
    weight = 1
    y = func()
    if(weight == 1){
      samples.add(y)
    }
  }
  return samples
}
```

Probabilistic programming 101: Rejection sampling

```
function threeDice(){
  x = rand(DiscreteUniform(1,6))
  y = rand(DiscreteUniform(1,6))
  z = rand(DiscreteUniform(1,6))
  observe(z == x + y)
  return x
}
samples = run(threeDice, 1000)
```



PPL implementation:

```
weight = 1
function observe(b){
  if(!b) weight = 0
}
function run(func, k){
  samples = []
  for(i in 1..k){
    weight = 1
    y = func()
    if(weight == 1){
      samples.add(y)
    }
  }
  return samples
}
```

Probabilistic programming 101: Importance sampling

```
function threeDice(){
  x = rand(DiscreteUniform(1,99))
  y = rand(DiscreteUniform(1,99))
  z = rand(DiscreteUniform(1,99))
  observe(z == x + y)
  return x
}
samples = run(threeDice, 1000)
```

Probabilistic programming 101: Importance sampling

```
function threeDice(){
  x = rand(DiscreteUniform(1,99))
  y = rand(DiscreteUniform(1,99))
  Z = DiscreteUniform(1,99)
  observe(Z, x + y)
  return x
}
samples = run(threeDice, 1000)
```

Probabilistic programming 101: Importance sampling

```
function threeDice(){
  x = rand(DiscreteUniform(1,99))
  y = rand(DiscreteUniform(1,99))
  Z = DiscreteUniform(1,99)
  observe(Z, x + y)
  return x
}
samples = run(threeDice, 1000)
```

PPL implementation:

```
weight = 1
function observe(D,x){
  weight *= prob(D,x)
}
function run(func, k){
  samples = []
  for(i in 1..k){
    weight = 1
    y = func()
    samples.add((weight,y))
  }
  return samples
}
```

Faster convergence

Probabilistic programming 101: Continuous distributions

Continuous distributions: `prob(D,x) = 0`.

- Rejection sampling rejects 100% of the trials
- Importance sampling only produces trials with `weight = 0`

Probabilistic programming 101: Continuous distributions

Continuous distributions: `prob(D,x) = 0`.

- Rejection sampling rejects 100% of the trials
- Importance sampling only produces trials with `weight = 0`

Standard solution: use probability density function `pdf(D,x)`:

```
function observe(D,x){ weight *= prob(D,x) }
```

$\downarrow$

```
function observe(D,x){ weight *= pdf(D,x) }
```

Probabilistic programming 101: Continuous distributions

Continuous distributions: `prob(D,x) = 0`.

- Rejection sampling rejects 100% of the trials
- Importance sampling only produces trials with `weight = 0`

Standard solution: use probability density function `pdf(D,x)`:

```
function observe(D,x){ weight *= prob(D,x) }
```

$$\downarrow$$

```
function observe(D,x){ weight *= pdf(D,x) }
```

Intuition: $\text{pdf}(D, x) \propto$ the probability that $\text{rand}(D)$ is close to x.

Probabilistic programming 101: Continuous distributions

Continuous distributions: `prob(D,x) = 0`.

- Rejection sampling rejects 100% of the trials
- Importance sampling only produces trials with `weight = 0`

Standard solution: use probability density function `pdf(D,x)`:

```
function observe(D,x){ weight *= prob(D,x) }
                                     ↓
function observe(D,x){ weight *= pdf(D,x) }
```

Intuition: $\text{pdf}(D, x) \propto$ the probability that $\text{rand}(D)$ is close to x.

Source of paradoxes

What went wrong: conditionals

Recall the drunk scientist:

```
if(flip(0.5)){
  observe(Normal(1.8, 0.5), h)
}else{
  observe(Normal(70, 10), w)
}
```

```
function observe(D,x){
  weight *= pdf(D,x)
}
```

What went wrong: conditionals

Recall the drunk scientist:

```
if(flip(0.5)){
  observe(Normal(1.8, 0.5), h)
}else{
  observe(Normal(70, 10), w)
}
```

```
function observe(D,x){
  weight *= pdf(D,x)
}
```

- **The PPL implementation is adding $m^{-1} + kg^{-1}$!**

$$\mathbb{E}[output] \approx \frac{\sum_{k=1}^{N}(weight_k) \cdot (output_k)}{\sum_{k=1}^{N}(weight_k)}$$

- The weight has units m^{-1} in some trials and kg^{-1} in other trials
- Probabilities don't have units, but pdf's do

Blame the programmer!

"It's your own responsibility to make the `weight` variable have consistent units."

Blame the programmer!

"It's your own responsibility to make the `weight` variable have consistent units."

- Semantics of `observe` = multiply `weight` by pdf
- Are we doing "accumulate a `weight`"-programming?
  - Pragmatist view
- Or are we doing **probabilistic** programming?
  - Purist view

What went wrong

Conditioning on events of measure zero is ambiguous!

What went wrong

Conditioning on events of measure zero is ambiguous!

$$A_\epsilon = \{(x, y) \in \mathbb{R}^2 : |x - y| \leq \epsilon\} \xrightarrow{\epsilon \to 0} \{(x, y) \in \mathbb{R}^2 : x = y\}$$

$$B_\epsilon = \{(x, y) \in \mathbb{R}^2 : |\exp(x) - \exp(y)| \leq \epsilon\} \xrightarrow{\epsilon \to 0} \{(x, y) \in \mathbb{R}^2 : x = y\}$$

What went wrong

Conditioning on events of measure zero is ambiguous!

$$A_\epsilon = \{(x,y) \in \mathbb{R}^2 : |x-y| \leq \epsilon\} \xrightarrow{\epsilon \to 0} \{(x,y) \in \mathbb{R}^2 : x = y\}$$

$$B_\epsilon = \{(x,y) \in \mathbb{R}^2 : |\exp(x) - \exp(y)| \leq \epsilon\} \xrightarrow{\epsilon \to 0} \{(x,y) \in \mathbb{R}^2 : x = y\}$$

"Although the sequences A_ϵ and B_ϵ tend to the same limit "$x = y$", the conditional densities $\mathbb{P}(x|A_\epsilon)$ and $\mathbb{P}(x|B_\epsilon)$ tend to different limits. As we see from this, merely to specify "$x = y$" without any qualifications is ambiguous. Whenever we have a probability density on one space and we wish to generate from it one on a subspace of measure zero, the only safe procedure is to pass to an explicitly defined limit by a process like A_ϵ and B_ϵ. In general, the final result will and must depend on which limiting operation was specified. This is extremely counter-intuitive at first hearing; yet it becomes obvious when the reason for it is understood."

– E.T. Jaynes (paraphrased)

Solution: don't condition on measure zero events

Problem: conditioning on events of measure zero is ambiguous.
Solution: condition on intervals.

```
observe(D, Interval(x,w))
```

Semantic meaning: rand(D) is in an interval of width w around x.

Solution: don't condition on measure zero events

Problem: conditioning on events of measure zero is ambiguous.
Solution: condition on intervals.

```
observe(D, Interval(x,w))
```

Semantic meaning: rand(D) is in an interval of width w around x.

Rejection sampling:

```
function observe(D,I){
  if(rand(D) not in I){ weight = 0 }
}
```

Solution: don't condition on measure zero events

Problem: conditioning on events of measure zero is ambiguous.
Solution: condition on intervals.

```
observe(D, Interval(x,w))
```

Semantic meaning: rand(D) is in an interval of width w around x.

Rejection sampling:

```
function observe(D,I){
  if(rand(D) not in I){ weight = 0 }
}
```

Importance sampling:

```
function observe(D,I){ weight *= probability(D,I) }
```

For intervals, `probability(D,I)` is nonzero.

Intervals remove unit anomalies

```
function centimeters(){
  h = rand(Normal(170, 50))
  if(flip(0.5)){
    observe(Normal(180, 10), Interval(h, 10))
  }
}
function meters(){
  h = rand(Normal(1.7, 0.5))
  if(flip(0.5)){
    observe(Normal(1.8, 0.1), Interval(h, 0.1))
  }
}
```

Intervals remove unit anomalies

```
function centimeters(){
  h = rand(Normal(170, 50))
  if(flip(0.5)){
    observe(Normal(180, 10), Interval(h, 10))
  }
}
function meters(){
  h = rand(Normal(1.7, 0.5))
  if(flip(0.5)){
    observe(Normal(1.8, 0.1), Interval(h, 0.1))
  }
}
```

- **Same output & no unit errors, even though observe is conditionally executed**
- **Rejection sampling and importance sampling converge to the same answer**

Take the limit

We still want to condition on measure zero events

Take the limit

We still want to condition on measure zero events

Idea: parameterize the program by the width of the interval, and take the limit $\textit{width} \to 0$

Take the limit

We still want to condition on measure zero events

Idea: parameterize the program by the width of the interval, and take the limit *width* $\to 0$

```
function drunk(width){
  h = rand(Normal(1.7, 0.5))
  w = rand(Normal(60, 10))
  if(flip(0.5)){
    observe(Normal(1.8, 0.1), Interval(h, A*width))
  }else{
    observe(Normal(70, 10), Interval(w, B*width))
  }
}
```

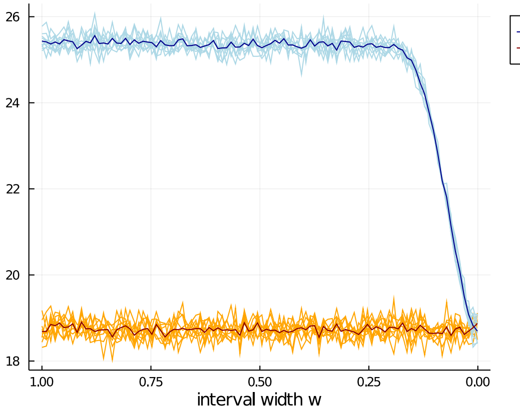
Since *width* **is unitless, we must introduce constants** A **and** B **with units** m **and** kg**.**
The relative size matters even as *width* $\to 0$**!**

Take the limit

We still want to condition on measure zero events

Idea: parameterize the program by the width of the interval, and take the limit $width \to 0$

```
function drunk(width){
  h = rand(Normal(1.7, 0.5))
  w = rand(Normal(60, 10))
  if(flip(0.5)){
    observe(Normal(1.8, 0.1), Int
  }else{
    observe(Normal(70, 10), Inter
  }
}
```



Since *width* **is unitless, we must introduce constants** A **and** B **with units** m **and** kg**.**

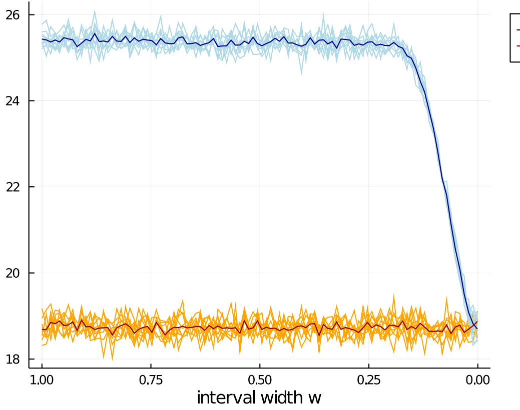
The relative size matters even as $width \to 0$**!**

Take the limit

We still want to condition on measure zero events

Idea: parameterize the program by the width of the interval, and take the limit *width* $\to 0$

```
function drunk(width){
  h = rand(Normal(1.7, 0.5))
  w = rand(Normal(60, 10))
  if(flip(0.5)){
    observe(Normal(1.8, 0.1), Int
  }else{
    observe(Normal(70, 10), Inter
  }
}
```



Since *width* is unitless, we must introduce constants A and B with units m and kg.
The relative size matters even as *width* $\to 0$!

Can we compute the limit $w \to 0$ directly?

Infinitesimal numbers

Definition

An infinitesimal number is a pair $(r, n) \in \mathbb{R} \times \mathbb{Z}$, which we write as $r\epsilon^n$.

Infinitesimal numbers

Definition

An infinitesimal number is a pair $(r, n) \in \mathbb{R} \times \mathbb{Z}$, which we write as $r\epsilon^n$.

$$r\epsilon^n \pm s\epsilon^k = \begin{cases} (r \pm s)\epsilon^n & \text{if } n = k \\ r\epsilon^n & \text{if } n < k \\ \pm s\epsilon^k & \text{if } n > k \end{cases}$$

$$(r\epsilon^n) \cdot (s\epsilon^k) = (r \cdot s)\epsilon^{n+k}$$

$$(r\epsilon^n)/(s\epsilon^k) = \begin{cases} (r/s)\epsilon^{n-k} & \text{if } s \neq 0 \\ \text{undefined} & \text{if } s = 0 \end{cases}$$

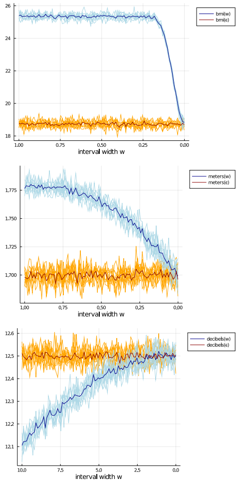
$$\text{probability}(D, \text{Interval}(x, r\epsilon^n)) = \begin{cases} \text{cdf}(D, x + \frac{1}{2}r) - \text{cdf}(D, x - \frac{1}{2}r) & \text{if } n = 0 \\ \text{pdf}(D, x) \cdot r\epsilon^n & \text{if } n > 0 \end{cases}$$

Infinitesimals give the limit

```
function bmi(width){
  h = rand(Normal(1.70, 0.2))
  w = rand(Normal(70, 30))
  if(flip(0.5)){
      observe(Normal(2.0,0.1), Interval(h,10*width))
  }else{
      observe(Normal(90,5), Interval(w,width))
  }
  return w / h^2
}

function meters(width){
  h = rand(Normal(1.7,0.5))
  if(flip(0.5)){
      observe(Normal(2.0,0.1), Interval(h,width))
  }
  return h
}

function decibels(width){
  x = rand(Normal(10,5))
  observe(Normal(15,5),Interval(x,width))
  return x
}
```



Consistency with non-zero width intervals:

`observe(D,Interval(x,eps))` gives the same result as

`observe(D,Interval(x,width))` and then taking the limit width $\to 0$

Parameter transformations

Intervals give reparameterisation invariance:

A function f maps Interval(x, ϵ) to Interval$(f(x), f'(x)\epsilon)$.

Parameter transformations

Intervals give reparameterisation invariance:

A function f maps Interval(x, ϵ) to Interval$(f(x), f'(x)\epsilon)$.

Original scale:

```
h = rand(Normal(1.7,0.5))
observe(Normal(1.8,0.5),
        Interval(h,eps))
return h
```

Answer: 1.75

Parameter transformations

Intervals give reparameterisation invariance:

A function f maps Interval(x, ϵ) to Interval$(f(x), f'(x)\epsilon)$.

Original scale:

```
h = rand(Normal(1.7,0.5))
observe(Normal(1.8,0.5),
        Interval(h,eps))
return h
```

Answer: 1.75

Logarithmic scale:

```
H = rand(LogNormal(1.7,0.5))
observe(LogNormal(1.8,0.5),
        exp(Interval(H,eps)))
return log(H)
```

Answer: 1.75

Same output $\Longrightarrow$ programs are invariant under choice of scale

(unit changes are a special case)

Recap

- Paradoxical behaviour
- Root of the problem: conditioning on measure-zero events is ambiguous
- Approach: rejection sampling as ground truth semantics
- Condition on intervals
- Measure-zero events as `Interval(x, eps)`
- Removes paradoxical behaviour: invariance under reparameterisations
- Proof of concept in Julia

Comments or questions?

mail@julesjacobs.com

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, Alex Lew, and Robbert Krebbers for help, inspiration, and discussions.