Trustworthy, Useful Languages for Probabilistic Modeling and Inference

Neil Toronto
Dissertation Defense
Brigham Young University

2014/06/11
Toronto et al. Super-Resolution via Recapture and Bayesian Effect Modeling. CVPR 2009
Master’s Research: Super-Resolution

• Model and query: Half a page of beautiful math

\[
\begin{align*}
C_{i,j}^x & \equiv i + \frac{1}{2} \quad i \in 0..m-1 \\
C_{i,j}^y & \equiv j + \frac{1}{2} \quad j \in 0..n-1 \\
N9(x, y) & \equiv \{i \in \mathbb{Z} \mid -1 \leq i - |x| \leq 1\} \\
& \times \{j \in \mathbb{Z} \mid -1 \leq j - |y| \leq 1\} \\
\text{dist}(x, y, \theta, d) & \equiv x \cos \theta + y \sin \theta - d \\
\text{prof}(d, \sigma, v^+, v^-) & \equiv \frac{v^+-v^-}{2} \text{erf} \left( \frac{d}{\sqrt{2}\sigma} \right) + \frac{v^++v^-}{2} \\
\text{edge}(x, y, \theta, d, v^+, v^-, \sigma) & \equiv \text{prof}(\text{dist}(x, y, \theta, d), \sigma, v^+, v^-) \\
S_{i,j}^{\text{edge}}(x, y) & \equiv \text{edge}(x-C_{i,j}^x, y-C_{i,j}^y, S_{i,j}^\theta, S_{i,j}^d, S_{i,j}^{v^+}, S_{i,j}^{v^-}, S_{i,j}^\sigma) \\
E[h(S_{x,y})] & \equiv \sum_{k,l \in N9(x,y)} w(x-C_{k,l}^x, y-C_{k,l}^y) h(S_{k,l}^{\text{edge}}(x, y)) \\
S_{i,j}^\theta & \sim \text{Uniform}(-\pi, \pi) \quad S_{i,j}^{v^+} \sim \text{Uniform}(0, 1) \\
S_{i,j}^d & \sim \text{Uniform}(-3, 3) \quad S_{i,j}^{v^-} \sim \text{Uniform}(0, 1) \\
S_{i,j}^\sigma & \sim \text{Beta}(1.6, 1) \\
I_{i,j} | S_{N9(i,j)} & \sim \text{Normal}(E[S_{i,j}], \omega) \\
\Phi_{i,j}(S_{N9(i,j)}) & \equiv \exp \left( -\frac{\text{Var}[S_{i,j}]}{2\gamma^2} \right)
\end{align*}
\]
Master’s Research: Super-Resolution

- Query implementation: 600 lines of Python
Main Results: Super-Resolution

• Competitor and BEI on 4x super-resolution:

Resolution Synthesis
Main Results: Super-Resolution

• Competitor and BEI on 4x super-resolution:
Main Results: Super-Resolution

• Competitor and BEI on 4x super-resolution:

Resolution Synthesis  Bayesian Edge Inference

• Beat state-of-the-art on “objective” measures
Main Results: Super-Resolution

• Competitor and BEI on 4x super-resolution:

  Resolution Synthesis  Bayesian Edge Inference

• Beat state-of-the-art on “objective” measures

• Was capable of other reconstruction tasks with few changes
Only Mostly Satisfying

Problem 1: Still not sure the program is right
Only Mostly Satisfying

Problem 1: Still not sure the program is right

Problem 2: *smooth* edges instead of *discontinuous*
Only Mostly Satisfying

Problem 1: Still not sure the program is right

Problem 2: *smooth* edges instead of *discontinuous*

“To approximate blurring with a spatially varying point-spread function (PSF), we assign each facet a Gaussian PSF and convolve each analytically *before combining outputs.*”
Problem 1: Still not sure the program is right

Problem 2: *smooth* edges instead of *discontinuous*

“To approximate blurring with a spatially varying point-spread function (PSF), we assign each facet a Gaussian PSF and convolve each analytically *before combining outputs.*”

i.e. “We can’t model it correctly so here’s a hack.”
Solution Idea: Probabilistic Language

```racket
#lang drbayes

(define (dist x y θ d)
  (- (+ (* x (cos θ))
       (* y (sin θ)))
    d))

(define (prof d σ v+ v-)
  (+ (* 1/2 (- v+ v-))
    (erf (/ d (* (sqrt 2) σ))))
  (+ (* 1/2 (+ v+ v-))))

(define (edge x y θ d v+ v-)
  (prof (dist x y θ d) σ v+ v-))

(define (scene-edge i j)
  (let ([θ (uniform (- pi) pi)]
        [d (uniform -3 3)]
        [σ (beta 1.6 1)]
        [v+ (uniform 0 1)]
        [v- (uniform 0 1)])
    (λ (x y)
      (edge (- x (- i 1/2)) (- y (+ j 1/2)) θ d v+ v-))))

(define (image-point i j)
  (normal (mean (map scene-edge (n9-x i j) (n9-y i j))) w))

(define (scene-reg i j)
  (exp (* -1/2 (/ (variance (map scene-edge (n9-x i j) (n9-y i j))) γ^2))))

(define (resize img)
  ....)
```

Solution Idea: Probabilistic Language

- Also somehow let me model correctly
Prior Work

Defined by an implementation

Defined by a semantics (i.e. mathematically)
Prior Work

Defined by an implementation

Defined by a semantics (i.e. mathematically)

Designed for Bayesian practice
Prior Work

Defined by an implementation

Defined by a semantics (i.e. mathematically)

Designed for Bayesian practice

Mimic human translation
Prior Work

Defined by an implementation

Defined by a semantics (i.e. mathematically)

Designed for Bayesian practice

Mimic human translation

Can’t tell error from feature
Prior Work

Defined by an implementation

Designed for Bayesian practice

Defined by a semantics (i.e. mathematically)

Mimic human translation

Can’t tell error from feature

Limited: usually no recursion or loops; conditions $X = c$
Prior Work

Defined by an implementation

Defined by a semantics (i.e. mathematically)

Designed for Bayesian practice

Designed for functional programmers or FP theorists

Mimic human translation

Can’t tell error from feature

Limited: usually no recursion or loops; conditions $X = c$
## Prior Work

<table>
<thead>
<tr>
<th>Defined by an implementation</th>
<th>Defined by a semantics (i.e. mathematically)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Designed for Bayesian practice</td>
<td>Designed for functional programmers or FP theorists</td>
</tr>
<tr>
<td>Mimic human translation</td>
<td>May not be implemented</td>
</tr>
<tr>
<td>Can’t tell error from feature</td>
<td></td>
</tr>
</tbody>
</table>

Limited: usually no recursion or loops; conditions $X = c$
### Prior Work

<table>
<thead>
<tr>
<th>Defined by an implementation</th>
<th>Defined by a semantics (i.e. mathematically)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Designed for Bayesian practice</td>
<td>Designed for functional programmers or FP theorists</td>
</tr>
<tr>
<td>Mimic human translation</td>
<td>May not be implemented</td>
</tr>
<tr>
<td>Can’t tell error from feature</td>
<td>Behavior is well-defined</td>
</tr>
<tr>
<td>Limited: usually no recursion or loops; conditions $X = c$</td>
<td></td>
</tr>
</tbody>
</table>
## Prior Work

<table>
<thead>
<tr>
<th>Defined by an implementation</th>
<th>Defined by a semantics (i.e. mathematically)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Designed for Bayesian practice</td>
<td>Designed for functional programmers or FP theorists</td>
</tr>
<tr>
<td>Mimic human translation</td>
<td>May not be implemented</td>
</tr>
<tr>
<td>Can’t tell error from feature</td>
<td>Behavior is well-defined</td>
</tr>
<tr>
<td>Limited: usually no recursion or loops; conditions $X = c$</td>
<td>Limited: usually finite distributions, no conditioning</td>
</tr>
</tbody>
</table>
## Prior Work

<table>
<thead>
<tr>
<th>Defined by an implementation</th>
<th>Defined by a semantics (i.e. mathematically)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Designed for Bayesian practice</td>
<td>Designed for functional programmers or FP theorists</td>
</tr>
<tr>
<td>Mimic human translation</td>
<td>May not be implemented</td>
</tr>
<tr>
<td>Can’t tell error from feature</td>
<td>Behavior is well-defined</td>
</tr>
<tr>
<td>Limited: usually no recursion or loops; conditions $X = c$</td>
<td>Limited: usually finite distributions, no conditioning</td>
</tr>
</tbody>
</table>

Best of all worlds: define language using functional programming theory, make it for Bayesians, and remove limitations
Thesis Statement

Functional programming theory and measure-theoretic probability provide a solid foundation for trustworthy, useful languages for constructive probabilistic modeling and inference.
Thesis Statement

Functional programming theory and measure-theoretic probability provide a solid foundation for trustworthy, useful languages for constructive probabilistic modeling and inference.

- **Useful**: let you think abstractly and handle details for you
Thesis Statement

Functional programming theory and measure-theoretic probability provide a solid foundation for trustworthy, useful languages for constructive probabilistic modeling and inference.

- Useful: let you think abstractly and handle details for you
- Trustworthy: defined mathematically
Thesis Statement

Functional programming theory and measure-theoretic probability provide a solid foundation for trustworthy, useful languages for constructive probabilistic modeling and inference.

• Useful: let you think abstractly and handle details for you

• Trustworthy: defined mathematically

• Functional programming theory has the tools to define programming languages mathematically
Thesis Statement

Functional programming theory and measure-theoretic probability provide a solid foundation for trustworthy, useful languages for constructive probabilistic modeling and inference.

- **Useful**: let you think abstractly and handle details for you
- **Trustworthy**: defined mathematically
- **Functional programming theory** has the tools to define programming languages mathematically
- **Measure-theoretic probability** is the most complete account of probability; should allow shedding common limitations
Simple Example Process

- Example process: Normal-Normal

\[ X \sim \text{Normal}(0, 1) \]
\[ Y \sim \text{Normal}(X, 1) \]
Simple Example Process

• Example process: Normal-Normal

\[ X \sim \text{Normal}(0, 1) \]
\[ Y \sim \text{Normal}(X, 1) \]

• Intuition: Sample \( X \), then sample \( Y \) using \( X \)
Simple Example Process

• Example process: Normal-Normal

\[ X \sim \text{Normal}(0, 1) \]
\[ Y \sim \text{Normal}(X, 1) \]

• Intuition: Sample \( X \), then sample \( Y \) using \( X \)

• Density model \( f : \mathbb{R} \times \mathbb{R} \rightarrow [0, \infty) \):
Simple Example Process

• Example process: Normal-Normal

\[ X \sim \text{Normal}(0, 1) \]
\[ Y \sim \text{Normal}(X, 1) \]

• Intuition: Sample \( X \), then sample \( Y \) using \( X \)

• Compute query \( \Pr[X < 0, Y \in (0, 2)] \) by integrating:
Conditional Queries

• Compute query $\Pr[X < 0 \mid Y = 1]$ using Bayes’ law:

$$f(x \mid y) = \frac{f(x, y)}{\int_{-\infty}^{\infty} f(x, y) \, dx}$$
Conditional Queries

- Compute query $\Pr[X < 0 \mid Y = 1]$ using Bayes’ law:

$$f(x \mid y) = \frac{f(x, y)}{\int_{-\infty}^{\infty} f(x, y) \, dx}$$
Conditional Queries

- Compute query $\Pr[X < 0 \mid Y = 1]$ using Bayes’ law:

$$f(x \mid y) = \frac{f(x, y)}{\int_{-\infty}^{\infty} f(x, y) \, dx}$$
Conditional Queries

• Compute query $\Pr[X < 0 \mid Y = 1]$ using Bayes’ law:

$$f(x \mid y) = \frac{f(x, y)}{\int_{-\infty}^{\infty} f(x, y) \, dx}$$
Conditional Queries

• Compute query $Pr[X < 0 \mid Y = 1]$ using Bayes’ law:

$$f(x \mid y) = \frac{f(x, y)}{\int_{-\infty}^{\infty} f(x, y) \, dx}$$

![Graph showing conditional probability distributions](image.png)
Conditional Queries

• Compute query $\Pr[X < 0 \mid Y = 1]$ using Bayes’ law:

$$f(x \mid y) = \frac{f(x, y)}{\int_{-\infty}^{\infty} f(x, y) \, dx}$$
So What Can’t Densities Model?
So What Can’t Densities Model?

• Tons of useful things that are easy to write down
So What Can’t Densities Model?

• Tons of useful things that are easy to write down
  ◦ Distributions given non-axial, zero-probability conditions

\[ X, Y \mid \sqrt{X^2 + Y^2} = 1 \]
So What Can’t Densities Model?

• Tons of useful things that are easy to write down
  
  ○ Distributions given non-axial, zero-probability conditions
    \[ X, Y \mid \sqrt{X^2 + Y^2} = 1 \]
  
  ○ Discontinuous change of variable (e.g. a thermometer)
    
    \[ Y \sim \text{Normal}(99, 1) \]
    \[ Y' = \min(100, Y) \]
So What Can’t Densities Model?

• Tons of useful things that are easy to write down

  ○ Distributions given non-axial, zero-probability conditions

    \[ X, Y \mid \sqrt{X^2 + Y^2} = 1 \]

  ○ Discontinuous change of variable (e.g. a thermometer)

    \[ Y \sim \text{Normal}(99, 1) \]
    \[ Y' = \min(100, Y) \]

  ○ Distributions of variable-dimension random variables

    \[ Z = \text{if } B \text{ then } \langle X_1, X_2 \rangle \text{ else } \langle X_1, X_2, X_3 \rangle \]
So What Can’t Densities Model?

• Tons of useful things that are easy to write down
  
  ○ Distributions given non-axial, zero-probability conditions
    \[ X, Y \mid \sqrt{X^2 + Y^2} = 1 \]
  
  ○ Discontinuous change of variable (e.g. a thermometer)
    \[ Y \sim \text{Normal}(99, 1) \]
    \[ Y' = \min(100, Y) \]
  
  ○ Distributions of variable-dimension random variables
    \[ Z = \text{if } B \text{ then } \langle X_1, X_2 \rangle \text{ else } \langle X_1, X_2, X_3 \rangle \]
  
  ○ Nontrivial distributions on infinite products
    \[ X_n \sim \text{Normal}(0, 1), \ n \in \mathbb{N} \]
So What Can’t Densities Model?

• Tons of useful things that are easy to write down
  ○ Distributions given non-axial, zero-probability conditions
    \[ X, Y \mid \sqrt{X^2 + Y^2} = 1 \]
  ○ Discontinuous change of variable (e.g. a thermometer)
    \[ Y \sim \text{Normal}(99, 1) \]
    \[ Y' = \min(100, Y) \]
  ○ Distributions of variable-dimension random variables
    \[ Z = \text{if } B \text{ then } \langle X_1, X_2 \rangle \text{ else } \langle X_1, X_2, X_3 \rangle \]
  ○ Nontrivial distributions on infinite products
    \[ X_n \sim \text{Normal}(0, 1), \ n \in \mathbb{N} \]
• Tricks to get around limitations aren’t general enough
Measure-Theoretic Probability

• Main ideas:
  ◦ Don’t assign *probability-like quantities* to *values*, assign *probabilities* to *sets* — *the probability query is king*
Measure-Theoretic Probability

• Main ideas:
  ◦ Don’t assign *probability-like quantities* to *values*, assign *probabilities* to *sets* — the probability query is king
  ◦ Confine assumed randomness to one place by making random variables *deterministic functions* that observe a random source
Measure-Theoretic Probability

• Main ideas:
  ◦ Don’t assign probability-like quantities to values, assign probabilities to sets — the probability query is king
  ◦ Confine assumed randomness to one place by making random variables deterministic functions that observe a random source

• Measure-theoretic model of example process:

\[
\Omega = \mathbb{R} \times \mathbb{R} \\
P : \text{Set}(\Omega) \rightarrow [0, 1], \quad P(A) = \int_A f \ d\lambda
\]
Measure-Theoretic Probability

• Main ideas:
  ◦ Don’t assign *probability-like quantities* to *values*, assign *probabilities* to *sets* — *the probability query is king*
  ◦ Confine assumed randomness to one place by making random variables *deterministic functions* that observe a random source

• Measure-theoretic model of example process:

\[
\Omega = \mathbb{R} \times \mathbb{R}
\]

\[
P : \text{Set}(\Omega) \rightarrow [0, 1], \quad P(A) = \int_A f \ d\lambda
\]

\[X : \Omega \rightarrow \mathbb{R}, \quad X(\omega) = \omega_0\]

\[Y : \Omega \rightarrow \mathbb{R}, \quad Y(\omega) = \omega_1\]
Measure-Theoretic Queries

• Specific query:

\[ \Pr[X < 0] = P(\{\omega \in \Omega \mid X(\omega) < 0\}) \]
Measure-Theoretic Queries

• Specific query:

\[ \Pr[X < 0] = P(\{\omega \in \Omega \mid X(\omega) < 0\}) \]

• Generalized:

\[ \Pr[Z \in C'] = P(\{\omega \in \Omega \mid Z(\omega) \in C'\}) \]
Measure-Theoretic Queries

• Specific query:

\[ \Pr[X < 0] = P(\{\omega \in \Omega \mid X(\omega) < 0\}) \]

• Generalized:

\[ \Pr[Z \in C] = P(\{\omega \in \Omega \mid Z(\omega) \in C\}) \]

• Conditional query: if \( \Pr[e_2] > 0 \) then

\[ \Pr[e_1 \mid e_2] = \frac{\Pr[e_1, e_2]}{\Pr[e_2]} \]
Measure-Theoretic Queries

• Specific query:
  \[ \Pr[X < 0] = P(\{\omega \in \Omega \mid X(\omega) < 0\}) \]

• Generalized:
  \[ \Pr[Z \in C] = P(\{\omega \in \Omega \mid Z(\omega) \in C\}) \]

• Conditional query: if \( \Pr[e_2] > 0 \) then
  \[ \Pr[e_1 \mid e_2] = \frac{\Pr[e_1, e_2]}{\Pr[e_2]} \]

Can we avoid densities when \( \Pr[e_2] = 0 \)?
Zero-Probability Conditions (Axial)

\[ \Pr[X < 0 \mid Y = 1] = \lim_{\varepsilon \to 0} \Pr[X < 0 \mid |Y - 1| < \varepsilon] \]
Zero-Probability Conditions (Axial)

\[
\Pr[X < 0 \mid Y = 1] = \lim_{\varepsilon \to 0} \frac{\Pr[X < 0 \mid |Y - 1| < \varepsilon]}{\Pr[|Y - 1| < \varepsilon]}
\]
Zero-Probability Conditions (Axial)

\[
\Pr[X < 0 \mid Y = 1] = \lim_{\varepsilon \to 0} \frac{\Pr[X < 0 \mid |Y - 1| < \varepsilon]}{\Pr[|Y - 1| < \varepsilon]}
\]

\[
= \lim_{\varepsilon \to 0} \frac{\Pr[X < 0, |Y - 1| < \varepsilon]}{\Pr[|Y - 1| < \varepsilon]}
\]
Zero-Probability Conditions (Axial)

\[
\Pr[X < 0 \mid Y = 1] = \lim_{\varepsilon \to 0} \frac{\Pr[X < 0 \mid |Y - 1| < \varepsilon]}{\Pr[|Y - 1| < \varepsilon]}
\]
Zero-Probability Conditions (Axial)

\[
\Pr[X < 0 \mid Y = 1] = \lim_{\varepsilon \to 0} \Pr[X < 0 \mid |Y - 1| < \varepsilon]
\]

\[= \lim_{\varepsilon \to 0} \frac{\Pr[X < 0, |Y - 1| < \varepsilon]}{\Pr[|Y - 1| < \varepsilon]}\]
Zero-Probability Conditions (Axial)

\[
\Pr[X < 0 \mid Y = 1] = \lim_{\varepsilon \to 0} \frac{\Pr[X < 0 \mid |Y - 1| < \varepsilon]}{\Pr[|Y - 1| < \varepsilon]}
\]
Zero-Probability Conditions (Axial)

\[
\Pr[X < 0 \mid Y = 1] = \lim_{\varepsilon \to 0} \frac{\Pr[X < 0 \mid |Y - 1| < \varepsilon]}{\Pr[|Y - 1| < \varepsilon]}
\]

= \lim_{\varepsilon \to 0} \frac{\Pr[X < 0, |Y - 1| < \varepsilon]}{\Pr[|Y - 1| < \varepsilon]}
Zero-Probability Conditions (Circular)

\[
\Pr[X < 0, Y < 0 \mid \sqrt{X^2 + Y^2} = 1] = \lim_{\varepsilon \to 0} \Pr[X < 0, Y < 0 \mid |\sqrt{X^2 + Y^2} - 1| < \varepsilon]
\]
Zero-Probability Conditions (Circular)

\[ \Pr[X < 0, Y < 0 \mid \sqrt{X^2 + Y^2} = 1] \]

\[ = \lim_{\varepsilon \to 0} \Pr[X < 0, Y < 0 \mid \sqrt{X^2 + Y^2} - 1 < \varepsilon] \]
Zero-Probability Conditions (Circular)

$$\Pr[X < 0, Y < 0 \mid \sqrt{X^2 + Y^2} = 1]$$

$$= \lim_{\varepsilon \to 0} \Pr[X < 0, Y < 0 \mid |\sqrt{X^2 + Y^2} - 1| < \varepsilon]$$
Zero-Probability Conditions (Circular)

\[
\Pr[X < 0, Y < 0 \mid \sqrt{X^2 + Y^2} = 1] = \lim_{\varepsilon \to 0} \Pr[X < 0, Y < 0 \mid |\sqrt{X^2 + Y^2} - 1| < \varepsilon]
\]
Zero-Probability Conditions (Circular)

\[ \Pr[X < 0, Y < 0 \mid \sqrt{X^2 + Y^2} = 1] = \lim_{\varepsilon \to 0} \Pr[X < 0, Y < 0 \mid |\sqrt{X^2 + Y^2} - 1| < \varepsilon] \]
Contribution: Don’t Integrate, Compute Backwards (1)

- Integration is hard!
Contribution: Don’t Integrate, Compute Backwards (1)

- Integration is hard!

- But random variables and $\Pr[\cdot]$ are an abstraction boundary hiding $\Omega$ and $P$, so we can choose convenient ones.
Contribution: Don’t Integrate, Compute Backwards (1)

- Integration is hard!

- But random variables and \( \Pr[\cdot] \) are an abstraction boundary hiding \( \Omega \) and \( P \), so we can choose convenient ones

A uniform random source model:

\[
\Omega = [0, 1] \times [0, 1] \\
P : \text{Set}(\Omega) \rightarrow [0, 1], \; P(A) = \lambda(A) \; \text{(i.e. } A\text{’s area)}
\]
Contribution: Don’t Integrate, Compute Backwards (1)

• Integration is hard!

• But random variables and $\Pr[\cdot]$ are an abstraction boundary hiding $\Omega$ and $P$, so we can choose convenient ones

A uniform random source model:

\[
\Omega = [0, 1] \times [0, 1] \\
P : \text{Set}(\Omega) \rightarrow [0, 1], \quad P(A) = \lambda(A) \quad \text{(i.e. A’s area)} \\
X : \Omega \rightarrow \mathbb{R}, \quad X(\omega) = F^{-1}(\omega_0) \\
Y : \Omega \rightarrow \mathbb{R}, \quad Y(\omega) = F^{-1}(\omega_1) + X(\omega)
\]

where $F : \mathbb{R} \rightarrow [0, 1]$ is the Normal CDF
Contribution: Don’t Integrate, Compute Backwards (1)

- Integration is hard!
- But random variables and \( \Pr[\cdot] \) are an abstraction boundary hiding \( \Omega \) and \( P \), so we can choose convenient ones

---

A uniform random source model:

\[
\begin{align*}
\Omega &= [0, 1] \times [0, 1] \\
P : \text{Set}(\Omega) &\rightarrow [0, 1], \ P(A) = \lambda(A) \quad \text{(i.e. \( A \)'s area)} \\
X : \Omega &\rightarrow \mathbb{R}, \ X(\omega) = F^{-1}(\omega_0) \\
Y : \Omega &\rightarrow \mathbb{R}, \ Y(\omega) = F^{-1}(\omega_1) + X(\omega)
\end{align*}
\]

where \( F : \mathbb{R} \rightarrow [0, 1] \) is the Normal CDF

- Stretches instead of integrates
Contribution: Don’t Integrate, Compute Backwards (2)

- Generalized query:

\[
\Pr[Z \in C] = P(\{\omega \in \Omega \mid Z(\omega) \in C\})
\]
Contribution: Don’t Integrate, Compute Backwards (2)

• Generalized query:

\[
\Pr[Z \in C] = P(\{\omega \in \Omega \mid Z(\omega) \in C\}) \\
= P(Z^{-1}(C))
\]

i.e. output distributions are defined by preimages
Contribution: Don’t Integrate, Compute Backwards (2)

• Generalized query:

\[
P_r[Z \in C] = P(\{\omega \in \Omega \mid Z(\omega) \in C\})
= P(Z^{-1}(C))
\]

i.e. output distributions are defined by **preimages**

• For a uniform random source model,

  ◦ Compute probabilities by computing **preimage** areas
Contribution: Don’t Integrate, Compute Backwards (2)

- Generalized query:

\[
\Pr[Z \in C] = P(\{\omega \in \Omega \mid Z(\omega) \in C\}) = P(Z^{-1}(C))
\]

i.e. output distributions are defined by preimages

- For a uniform random source model,
  - Compute probabilities by computing preimage areas
  - Compute conditional probabilities as quotients of preimage areas
Contribution: Don’t Integrate, Compute Backwards (2)

• Generalized query:

\[ P[Z \in C] = P(\{\omega \in \Omega \mid Z(\omega) \in C\}) = P(Z^{-1}(C)) \]

i.e. output distributions are defined by preimages

• For a uniform random source model,

  ◦ Compute probabilities by computing preimage areas

  ◦ Compute conditional probabilities as quotients of preimage areas

• Is this really more feasible than integrating?
Queries Using Preimages

\[ \Pr[X < 0, Y < 0 \mid \sqrt{X^2 + Y^2} = 1] = \lim_{\varepsilon \to 0} \Pr[X < 0, Y < 0 \mid |\sqrt{X^2 + Y^2} - 1| < \varepsilon] \]
Queries Using Preimages

$$\Pr[X < 0, Y < 0 | \sqrt{X^2 + Y^2} = 1]$$

$$= \lim_{\varepsilon \to 0} \Pr[X < 0, Y < 0 | |\sqrt{X^2 + Y^2} - 1| < \varepsilon]$$

Uniform Random Source

Original Model
Queries Using Preimages

\[ \Pr[X < 0, Y < 0 \mid \sqrt{X^2 + Y^2} = 1] \]

\[ = \lim_{\varepsilon \to 0} \Pr[X < 0, Y < 0 \mid |\sqrt{X^2 + Y^2} - 1| < \varepsilon] \]

Uniform Random Source

Original Model
Queries Using Preimages

\[
\Pr[X < 0, Y < 0 \mid \sqrt{X^2 + Y^2} = 1]
\]

\[
= \lim_{\varepsilon \to 0} \Pr[X < 0, Y < 0 \mid \left| \sqrt{X^2 + Y^2} - 1 \right| < \varepsilon]
\]

Uniform Random Source

Original Model
Queries Using Preimages

\[
\Pr[X < 0, Y < 0 \mid \sqrt{X^2 + Y^2} = 1] = \lim_{\varepsilon \to 0} \Pr[X < 0, Y < 0 \mid |\sqrt{X^2 + Y^2} - 1| < \varepsilon]
\]
Crazy Idea is Feasible If...

• Seems like we need:
  ◦ Standard interpretation of programs as pure functions from a random source
Crazy Idea is Feasible If...

• Seems like we need:
  ◦ Standard interpretation of programs as pure functions from a random source
  ◦ Efficient way to compute preimage sets
Crazy Idea is Feasible If...

• Seems like we need:
  ◦ Standard interpretation of programs as pure functions from a random source
  ◦ Efficient way to compute preimage sets
  ◦ Efficient representation of arbitrary sets
Crazy Idea is Feasible If...

- Seems like we need:
  - Standard interpretation of programs as pure functions from a random source
  - Efficient way to compute preimage sets
  - Efficient representation of arbitrary sets
  - Efficient way to compute areas of preimage sets
Crazy Idea is Feasible If...

- Seems like we need:
  - Standard interpretation of programs as pure functions from a random source
  - Efficient way to compute preimage sets
  - Efficient representation of arbitrary sets
  - Efficient way to compute areas of preimage sets
  - Proof of correctness w.r.t. standard interpretation
Crazy Idea is Feasible If...

- Seems like we need:
  - Standard interpretation of programs as pure functions from a random source
  - Efficient way to compute preimage sets
  - Efficient representation of arbitrary sets
  - Efficient way to compute areas of preimage sets
  - Proof of correctness w.r.t. standard interpretation

- Completely infeasible! But...
What About Approximating?

Conservative approximation with rectangles:
What About Approximating?

Conservative approximation with rectangles:
What About Approximating?

Restricting preimages to rectangular subdomains:
What About Approximating?

Restricting preimages to rectangular subdomains:
What About Approximating?

Restricting preimages to rectangular subdomains:
What About Approximating?

Restricting preimages to rectangular subdomains:
What About Approximating?

Restricting preimages to rectangular subdomains:
What About Approximating?

Restricting preimages to rectangular subdomains:
What About Approximating?

Restricting preimages to rectangular subdomains:
What About Approximating?

Restricting preimages to rectangular subdomains:
What About Approximating?

Restricting preimages to rectangular subdomains:
What About Approximating?

Restricting preimages to rectangular subdomains:
What About Approximating?

Restricting preimages to rectangular subdomains:
What About Approximating?

Restricting preimages to rectangular subdomains:
What About Approximating?

Sampling: exponential to quadratic (e.g. days to minutes)
What About Approximating?

Sampling: exponential to quadratic (e.g. days to minutes)
Crazy Idea is Actually Feasible If...

• Standard interpretation of programs as pure functions from a random source

• Efficient way to compute preimage sets

• Efficient representation of arbitrary sets

• Efficient way to compute volumes of preimage sets

• Proof of correctness w.r.t. standard interpretation
Crazy Idea is Actually Feasible If...

- Standard interpretation of programs as pure functions from a random source
- Efficient way to compute approximate preimage subsets
- Efficient representation of arbitrary sets
- Efficient way to compute volumes of preimage sets

- Proof of correctness w.r.t. standard interpretation
Crazy Idea is Actually Feasible If...

• Standard interpretation of programs as pure functions from a random source

• Efficient way to compute approximate preimage subsets

• Efficient representation of approximating sets

• Efficient way to compute volumes of preimage sets

• Proof of correctness w.r.t. standard interpretation
Crazy Idea is Actually Feasible If...

- Standard interpretation of programs as pure functions from a random source
- Efficient way to compute approximate preimage subsets
- Efficient representation of approximating sets
- Efficient way to sample uniformly in preimage sets

- Proof of correctness w.r.t. standard interpretation
Crazy Idea is Actually Feasible If...

• Standard interpretation of programs as pure functions from a random source

• Efficient way to compute approximate preimage subsets

• Efficient representation of approximating sets

• Efficient way to sample uniformly in preimage sets
  ○ Efficient domain partition sampling

• Proof of correctness w.r.t. standard interpretation
Crazy Idea is Actually Feasible If...

- Standard interpretation of programs as pure functions from a random source
- Efficient way to compute approximate preimage subsets
- Efficient representation of approximating sets
- Efficient way to sample uniformly in preimage sets
  - Efficient domain partition sampling
  - Efficient way to determine whether a domain sample is actually in the preimage (just use standard interpretation)
- Proof of correctness w.r.t. standard interpretation
Standard Interpretation

• Grammar:

\[
p ::= x := e; \cdots; x := e; e
\]
\[
e ::= x e | \text{if } e e e | \text{let } e e | \text{env } n | \langle e, e \rangle | \delta e | v
\]
\[
x ::= \text{[first-order function names]}
\]
\[
\delta ::= \text{[primitive function names]}
\]
\[
v ::= \text{[first-order values]}
\]
Standard Interpretation

• Grammar:

\[
p ::= x := e; \; \ldots \; ; x := e; \; e \\
 e ::= x \; e | \text{if} \; e \; e \; e | \text{let} \; e \; e | \text{env} \; n | \langle e, e \rangle | \delta \; e | v \\
x ::= \text{[first-order function names]} \\
\delta ::= \text{[primitive function names]} \\
v ::= \text{[first-order values]} \\
\]

• Semantic function $[] : p \rightarrow (\Omega \rightarrow B)$
Standard Interpretation

• Grammar:

\[
p ::= x := e ; \cdots ; x := e ; e
\]

\[e ::= x e | \text{if } e e e | \text{let } e e | \text{env } n | \langle e, e \rangle | \delta e | v\]

\[x ::= \text{[first-order function names]}\]

\[\delta ::= \text{[primitive function names]}\]

\[v ::= \text{[first-order values]}\]

• Semantic function \([\cdot] : p \rightarrow (\Omega \rightarrow B)\)

• Math has no general recursion, so \([p]\) (i.e. interpretation of program \(p\)) is a \(\lambda\)-calculus term
Standard Interpretation

- Grammar:

\[
p ::= x := e ; \cdots ; x := e ; e \\
\]

\[
e ::= x e | \text{if } e e e | \text{let } e e | \text{env } n | \langle e, e \rangle | \delta e | v \\
\]

\[
x ::= \text{[first-order function names]} \\
\delta ::= \text{[primitive function names]} \\
v ::= \text{[first-order values]} \\
\]

- Semantic function \([\cdot] : p \rightarrow (\Omega \rightarrow B)\)

- Math has no general recursion, so \([p] \) (i.e. interpretation of program \(p\)) is a \(\lambda\)-calculus term

- Easy implementation in any language with lambdas
Compositional Semantics

- **Compositional**: every term’s meaning depends only on its immediate subterms’ meanings
Compositional Semantics

• **Compositional**: every term’s meaning depends only on its immediate subterms’ meanings

• Advantage: proofs about all programs by structural induction
• **Compositional**: every term’s meaning depends only on its immediate subterms’ meanings

• Advantage: proofs about all programs by structural induction

• Example: meaning of \( \langle x + y, x \cdot y \rangle \)
Compositional Semantics

- **Compositional**: every term’s meaning depends only on its immediate subterms’ meanings

- Advantage: proofs about all programs by structural induction

- Example: meaning of $\langle x + y, x \cdot y \rangle$

$$\llbracket \langle x + y, x \cdot y \rangle \rrbracket = \text{pair } \llbracket x + y \rrbracket \llbracket x \cdot y \rrbracket$$
Compositional Semantics

- **Compositional**: every term’s meaning depends only on its immediate subterms’ meanings

- Advantage: proofs about all programs by structural induction

- Example: meaning of \( \langle x + y, x \cdot y \rangle \)

  \[
  \left[ \langle x + y, x \cdot y \rangle \right] = \text{pair} \left[ x + y \right] \left[ x \cdot y \right]
  \]

- Nonexample:

  \[
  \left[ \langle x + y, x \cdot y \rangle \right] = \text{pair} \left( \text{plus} \left[ x \right] \left[ y \right] \right) \left[ x \cdot y \right]
  \]
Compositional Semantics

• **Compositional**: every term’s meaning depends only on its immediate subterms’ meanings

• Advantage: proofs about all programs by structural induction

• Example: meaning of $\langle x + y, x \cdot y \rangle$

$$\llbracket \langle x + y, x \cdot y \rangle \rrbracket = \text{pair } \llbracket x + y \rrbracket \llbracket x \cdot y \rrbracket$$

• Nonexample:

$$\llbracket \langle x + y, x \cdot y \rangle \rrbracket = \text{pair } (\text{plus } \llbracket x \rrbracket \llbracket y \rrbracket) \llbracket x \cdot y \rrbracket$$

pair : $(A \rightarrow B_1) \rightarrow (A \rightarrow B_2) \rightarrow (A \rightarrow \langle B_1, B_2 \rangle)$

pair $f_1 \ f_2 = \lambda a. \langle f_1 \ a, f_2 \ a \rangle$
Compositional Semantics

- **Compositional**: every term’s meaning depends only on its immediate subterms’ meanings

- Advantage: proofs about all programs by structural induction

- Example: meaning of \( \langle x + y, x \cdot y \rangle \)

\[
\llbracket \langle x + y, x \cdot y \rangle \rrbracket = \text{pair } \llbracket x + y \rrbracket \llbracket x \cdot y \rrbracket
\]

- Nonexample:

\[
\llbracket \langle x + y, x \cdot y \rangle \rrbracket = \text{pair } ( \text{plus } \llbracket x \rrbracket \llbracket y \rrbracket ) \llbracket x \cdot y \rrbracket
\]

\[
\text{pair} : (A \rightarrow B_1) \rightarrow (A \rightarrow B_2) \rightarrow (A \rightarrow \langle B_1, B_2 \rangle)
\]

\[
\text{pair } f_1 \ f_2 = \lambda a. \langle f_1 \ a, f_2 \ a \rangle
\]

- Can preimages be computed compositionally?
Pair Preimages

\[ f_1(\omega) = \omega_0 + \omega_1 \quad f_2(\omega) = \omega_0 \cdot \omega_1 \]
Pair Preimages

\[ f_1(\omega) = \omega_0 + \omega_1 \quad f_2(\omega) = \omega_0 \cdot \omega_1 \]

\[ f = \text{pair } f_1 \ f_2 = \lambda \omega. \langle \omega_0 + \omega_1, \omega_0 \cdot \omega_1 \rangle \]
Pair Preimages

\[ f_1(\omega) = \omega_0 + \omega_1 \quad f_2(\omega) = \omega_0 \cdot \omega_1 \]

\[ f = \text{pair } f_1 \ f_2 = \lambda \omega. \langle \omega_0 + \omega_1, \omega_0 \cdot \omega_1 \rangle \]

\[ f_1^{-1}([0.5, 0.7]) : \]
Pair Preimages

\[ f_1(\omega) = \omega_0 + \omega_1 \quad f_2(\omega) = \omega_0 \cdot \omega_1 \]

\[ f = \text{pair } f_1 \ f_2 = \lambda \omega. \langle \omega_0 + \omega_1, \omega_0 \cdot \omega_1 \rangle \]

\[ f_1^{-1}([0.5, 0.7]) \text{ and } f_2^{-1}([0.05, 0.1]) : \]
Pair Preimages

\[ f_1(\omega) = \omega_0 + \omega_1 \quad f_2(\omega) = \omega_0 \cdot \omega_1 \]

\[ f = \text{pair } f_1 \ f_2 = \lambda \omega. \langle \omega_0 + \omega_1, \omega_0 \cdot \omega_1 \rangle \]

\( f^{-1}([0.5, 0.7] \times [0.05, 0.1]) \):
Nonstandard Interpretation: Computing Preimages

- Preimage computation:
Nonstandard Interpretation: Computing Preimages

- Preimage computation:

\[
A \overset{\text{pre}}{\sim} B \::= \text{Set}(A) \rightarrow \langle \text{Set}(B), \text{Set}(B) \rightarrow \text{Set}(A) \rangle
\]
Nonstandard Interpretation: Preimages Under Pairing

• Pairing types:

\[
\text{pair} : (A \to B_1) \to (A \to B_2) \to (A \to \langle B_1, B_2 \rangle)
\]

\[
\text{pair}_{\text{pre}} : (A \pre B_1) \to (A \pre B_2) \to (A \pre \langle B_1, B_2 \rangle)
\]
Nonstandard Interpretation: Preimages Under Pairing

- Pairing types:

\[
\text{pair} : (A \rightarrow B_1) \rightarrow (A \rightarrow B_2) \rightarrow (A \rightarrow \langle B_1, B_2 \rangle) \\
\text{pair}_{\text{pre}} : (A \premapsto B_1) \rightarrow (A \premapsto B_2) \rightarrow (A \premapsto \langle B_1, B_2 \rangle)
\]

Theorem (correctness under pairing). If

\[
\circ h_1 : A \premapsto B_1 \text{ computes preimages under } f_1 : A \rightarrow B_1
\]
Nonstandard Interpretation: Preimages Under Pairing

• Pairing types:

\[
\text{pair} : (A \to B_1) \to (A \to B_2) \to (A \to \langle B_1, B_2 \rangle)
\]

\[
\text{pair}_{\text{pre}} : (A \overset{\text{pre}}{\to} B_1) \to (A \overset{\text{pre}}{\to} B_2) \to (A \overset{\text{pre}}{\to} \langle B_1, B_2 \rangle)
\]

Theorem (correctness under pairing). If

\( h_1 : A \overset{\text{pre}}{\to} B_1 \) computes preimages under \( f_1 : A \to B_1 \)

\( h_2 : A \overset{\text{pre}}{\to} B_2 \) computes preimages under \( f_2 : A \to B_2 \)
Nonstandard Interpretation: Preimages Under Pairing

- Pairing types:

\[
\text{pair} : (A \rightarrow B_1) \rightarrow (A \rightarrow B_2) \rightarrow (A \rightarrow \langle B_1, B_2 \rangle)
\]

\[
\text{pair}_{\text{pre}} : (A \overset{\text{pre}}{\rightarrow} B_1) \rightarrow (A \overset{\text{pre}}{\rightarrow} B_2) \rightarrow (A \overset{\text{pre}}{\rightarrow} \langle B_1, B_2 \rangle)
\]

**Theorem (correctness under pairing).** If

- \( h_1 : A \overset{\text{pre}}{\rightarrow} B_1 \) computes preimages under \( f_1 : A \rightarrow B_1 \)

- \( h_2 : A \overset{\text{pre}}{\rightarrow} B_2 \) computes preimages under \( f_2 : A \rightarrow B_2 \)

then \( \text{pair}_{\text{pre}} h_1 h_2 \) computes preimages under \( \text{pair} f_1 f_2 \).
Nonstandard Interpretation: Preimages Under Pairing

• Pairing types:

\[
pair : (A \rightarrow B_1) \rightarrow (A \rightarrow B_2) \rightarrow (A \rightarrow \langle B_1, B_2 \rangle)
\]
\[
pair_{\text{pre}} : (A \overset{\text{pre}}{\rightarrow} B_1) \rightarrow (A \overset{\text{pre}}{\rightarrow} B_2) \rightarrow (A \overset{\text{pre}}{\rightarrow} \langle B_1, B_2 \rangle)
\]

**Theorem (correctness under pairing).** If

- \( h_1 : A \overset{\text{pre}}{\rightarrow} B_1 \) computes preimages under \( f_1 : A \rightarrow B_1 \)
- \( h_2 : A \overset{\text{pre}}{\rightarrow} B_2 \) computes preimages under \( f_2 : A \rightarrow B_2 \)

then \( \text{pair}_{\text{pre}} h_1 h_2 \) computes preimages under \( \text{pair} f_1 f_2 \).

*Proof sketch.* Preimages distribute over cartesian products.
Nonstandard Interpretation: Preimages Under Pairing

• Pairing types:

\[
\text{pair} : (A \to B_1) \to (A \to B_2) \to (A \to \langle B_1, B_2 \rangle)
\]
\[
\text{pair}_{\text{pre}} : (A \overset{\text{pre}}{\to} B_1) \to (A \overset{\text{pre}}{\to} B_2) \to (A \overset{\text{pre}}{\to} \langle B_1, B_2 \rangle)
\]

Theorem (correctness under pairing). If

\[
\circ h_1 : A \overset{\text{pre}}{\to} B_1 \text{ computes preimages under } f_1 : A \to B_1
\]
\[
\circ h_2 : A \overset{\text{pre}}{\to} B_2 \text{ computes preimages under } f_2 : A \to B_2
\]

then \( \text{pair}_{\text{pre}} h_1 h_2 \) computes preimages under \( \text{pair } f_1 f_2 \).

Proof sketch. Preimages distribute over cartesian products.

• Similar theorems for every kind of term
Theorem. For all programs $p$, $\llbracket p \rrbracket_{\text{pre}}$ computes preimages under $\llbracket p \rrbracket$.

Proof. By structural induction on program terms.
• Q. Don’t the interpretations of $[\cdot]_{\text{pre}}$ do uncountable things?
Wait a Minute

• Q. Don’t the interpretations of $[\cdot]_{\text{pre}}$ do uncountable things?
  
  ○ A. Yes. Yes, they do.
• Q. Don’t the interpretations of $[\cdot]_{\text{pre}}$ do uncountable things?
  ○ A. Yes. Yes, they do.
• Q. Where do I get a computer that runs them?
• Q. Don’t the interpretations of $[\cdot]_\text{pre}$ do uncountable things?
  ○ A. Yes. Yes, they do.

• Q. Where do I get a computer that runs them?
  ○ A. Nowhere, but we’ll approximate them soon.
Wait a Minute

• Q. Don’t the interpretations of $\cdot_{pre}$ do uncountable things?
  ○ A. Yes. Yes, they do.

• Q. Where do I get a computer that runs them?
  ○ A. Nowhere, but we’ll approximate them soon.

• Q. Why interpret programs as uncomputable functions, then?
Wait a Minute

- Q. Don’t the interpretations of $\cdot \mathcal{P} \cdot$ do uncountable things?
  - A. Yes. Yes, they do.

- Q. Where do I get a computer that runs them?
  - A. Nowhere, but we’ll approximate them soon.

- Q. Why interpret programs as uncomputable functions, then?
  - A. So we know exactly what to approximate.
Wait a Minute

• Q. Don’t the interpretations of \([\cdot]_{\text{pre}}\) do uncountable things?
  ○ A. Yes. Yes, they do.

• Q. Where do I get a computer that runs them?
  ○ A. Nowhere, but we’ll approximate them soon.

• Q. Why interpret programs as uncomputable functions, then?
  ○ A. So we know exactly what to approximate.

• Q. Where did you get a \(\lambda\)-calculus that could operate on arbitrary, possibly infinite sets, anyway?
  ○ A. Well...
Lambda-ZFC

\[ \text{\textlambda \ text{calculus}} \]
Lambda-ZFC

\[ \lambda \text{ calculus} + \text{ Infinite sets and operations} \]
Lambda-ZFC

$\lambda$ calculus + Infinite sets and operations = $\lambda_{ZFC}$
Lambda-ZFC

- Contemporary math, but with lambdas and general recursion; or functional programming, but with infinite sets
Lambda-ZFC

• Contemporary math, but with lambdas and general recursion; or functional programming, but with infinite sets

• Can express uncountably infinite operations, can’t solve its own halting problem


Lambda-ZFC

- Contemporary math, but with lambdas and general recursion; or functional programming, but with infinite sets
- Can express uncountably infinite operations, can’t solve its own halting problem
- Can use contemporary mathematical theorems directly
Rectangular Approximation

• A rectangle is
  ◦ An interval or union of intervals
  ◦ $A \times B$ for rectangles $A$ and $B$
Rectangular Approximation

• A rectangle is
  ° An interval or union of intervals
  ° $A \times B$ for rectangles $A$ and $B$

• Easy representation; easy intersection and join (union-like) operation, empty test, other operations
Rectangular Approximation

- A rectangle is
  - An interval or union of intervals
  - $A \times B$ for rectangles $A$ and $B$

- Easy representation; easy intersection and join (union-like) operation, empty test, other operations

- Recall:
  $$A \xrightarrow{\text{pre}} B ::= \text{Set}(A) \rightarrow \langle \text{Set}(B), \text{Set}(B) \rightarrow \text{Set}(A) \rangle$$
Rectangular Approximation

- A rectangle is
  - An interval or union of intervals
  - $A \times B$ for rectangles $A$ and $B$

- Easy representation; easy intersection and join (union-like) operation, empty test, other operations

- Recall:
  $$A \preceq B ::= \text{Set}(A) \rightarrow \langle \text{Set}(B), \text{Set}(B) \rightarrow \text{Set}(A) \rangle$$

- Define:
  $$A \preceq' B ::= \text{Rect}(A) \rightarrow \langle \text{Rect}(B), \text{Rect}(B) \rightarrow \text{Rect}(A) \rangle$$
Rectangular Approximation

• A rectangle is
  
  ◦ An interval or union of intervals
  
  ◦ $A \times B$ for rectangles $A$ and $B$

• Easy representation; easy intersection and join (union-like) operation, empty test, other operations

• Recall:

\[
A \xrightarrow{\text{pre}} B ::= \text{Set}(A) \rightarrow \langle \text{Set}(B), \text{Set}(B) \rightarrow \text{Set}(A) \rangle
\]

• Define:

\[
A \xrightarrow{\text{pre}} B' ::= \text{Rect}(A) \rightarrow \langle \text{Rect}(B), \text{Rect}(B) \rightarrow \text{Rect}(A) \rangle
\]

• Derive $[\cdot]_{\text{pre}}' : p \rightarrow (\Omega \xrightarrow{\text{pre}} B)$
In Theory...

**Theorem (sound).** $\lbrack \cdot \rbrack_{\text{pre}}'$ computes overapproximations of the preimages computed by $\lbrack \cdot \rbrack_{\text{pre}}$.

- Consequence: Sampling within preimages doesn’t leave anything out
Theorem (sound). $\left[ \cdot \right]_{\text{pre}}'$ computes overapproximations of the preimages computed by $\left[ \cdot \right]_{\text{pre}}$.

- Consequence: Sampling within preimages doesn’t leave anything out

Theorem (monotone). $\left[ \cdot \right]_{\text{pre}}'$ is monotone.

- Consequence: Partitioning the domain never increases approximate preimages
In Theory...

Theorem (sound). $\mathcal{P}_\text{pre}$ computes overapproximations of the preimages computed by $\mathcal{P}_\text{pre}$.

• Consequence: Sampling within preimages doesn’t leave anything out

Theorem (monotone). $\mathcal{P}_\text{pre}$ is monotone.

• Consequence: Partitioning the domain never increases approximate preimages

Theorem (decreasing). $\mathcal{P}_\text{pre}$ never returns preimages larger than the given subdomain.

• Consequence: Refining preimage partitions never explodes
In Practice...

Theorems prove this always works:
In Practice...

Theorems prove this always works:
In Practice...

Theorems prove this always works:
In Practice...

Theorems prove this always works:
Theorems prove this always works:
Theorems prove this always works:
In Practice...

Theorems prove this always works:
In Practice...

Theorems prove this always works:
In Practice...

Theorems prove this always works:
Importance Sampling

• Alternative to arbitrarily low-rate rejection sampling:
Importance Sampling

- Alternative to arbitrarily low-rate rejection sampling:

First, refine using preimage computation:
Importance Sampling

• Alternative to arbitrarily low-rate rejection sampling:

Second, randomly choose from arbitrarily fine partition:
Importance Sampling

• Alternative to arbitrarily low-rate rejection sampling:

Third, refine again:
Importance Sampling

- Alternative to arbitrarily low-rate rejection sampling:

Fourth, sample uniformly:
Importance Sampling

• Alternative to arbitrarily low-rate rejection sampling:

Do process “in the limit”; i.e. choose $[\omega_0, \omega_0] \times \Omega_1$: 

\[
1 \quad 0.8 \quad 0.6 \quad 0.4 \quad 0.2 \quad 0
\]

\[
0 \quad 0.2 \quad 0.4 \quad 0.6 \quad 0.8 \quad 1
\]
What About Recursion?

• General recursion, programs that halt with probability 1; e.g.

```
(define/drbayes (geometric p)
 (if (bernoulli p)
     0
     (+ 1 (geometric p)))))
```
What About Recursion?

• General recursion, programs that halt with probability 1; e.g.

\[
\text{(define/drbayes (geometric p)}
\text{ (if (bernoulli p)}
\text{ 0}
\text{ (+ 1 (geometric p)))))}
\]

• Consider programs as being fully inlined (thus infinite):

\[
\text{(if (bernoulli p)}
\text{ 0}
\text{ (+ 1 (if (bernoulli p)}
\text{ 0}
\text{ (+ 1 ...))))))}
\]
What About Recursion?

- General recursion, programs that halt with probability 1; e.g.

  (define/drbayes (geometric p)
    (if (bernoulli p)
      0
      (+ 1 (geometric p)))))

- Consider programs as being fully inlined (thus infinite):

  (if (bernoulli p)
    0
    (+ 1 (if (bernoulli p)
      0
      (+ 1 (if (bernoulli p)
      0
      (+ 1 ...))))))

- Random domain needs to be big enough and the right shape
Program Domain Values

• Values $\omega \in \Omega$ are infinite binary trees:
Program Domain Values

- Values $\omega \in \Omega$ are infinite binary trees:

```
  0.12351...
  /     \     
0.52198...  0.33780...
/  \           /  
0.92462...  0.52309...  0.00143...  0.99264...
```

- Every expression in a program is assigned a node
Program Domain Values

- Values $\omega \in \Omega$ are infinite binary trees:

- Every expression in a program is assigned a node
- Implemented using lazy trees of random values
• Values $\omega \in \Omega$ are infinite binary trees:

• Every expression in a program is assigned a node

• Implemented using lazy trees of random values

• No probability density for domain, but there is a measure
Demo: Normal-Normal With Circular Condition

- Normal-Normal process:

  \[ X \sim \text{Normal}(0, 1) \]
  \[ Y \sim \text{Normal}(X, 1) \]
Demo: Normal-Normal With Circular Condition

• Normal-Normal process:

\[ X \sim \text{Normal}(0, 1) \]
\[ Y \sim \text{Normal}(X, 1) \]

• Objective: Find the distribution of \( X, Y \mid \sqrt{X^2 + Y^2} = 1 \)
Demo: Normal-Normal With Circular Condition

- Normal-Normal process:
  \[ X \sim \text{Normal}(0, 1) \]
  \[ Y \sim \text{Normal}(X, 1) \]

- Objective: Find the distribution of \( X, Y | \sqrt{X^2 + Y^2} = 1 \)

- Implementation:

  `(define/drbayes e
   (let* ([x (normal 0 1)]
          [y (normal x 1)])
      (list x y (sqrt (+ (sqr x) (sqr y))))))`
Demo: Normal-Normal With Circular Condition

- Normal-Normal process:

\[
X \sim \text{Normal}(0, 1) \\
Y \sim \text{Normal}(X, 1)
\]

- Objective: Find the distribution of \( X, Y \mid \sqrt{X^2 + Y^2} = 1 \)

- Implementation:

```
(define/drbayes e
  (let* ([x (normal 0 1)]
         [y (normal x 1)])
    (list x y (sqrt (+ (sqr x) (sqr y)))))))
```

- Goal: Sample in the preimage of

```
(set-list reals reals (interval (- 1 ε) (+ 1 ε)))
```
Demo: Normal-Normal With Circular Condition

For $\varepsilon = 0.01$:

Preimage rectangles
Demo: Normal-Normal With Circular Condition

For $\varepsilon = 0.01$: 

![Graph showing preimage samples](image-url)
Demo: Normal-Normal With Circular Condition

For $\varepsilon = 0.01$:

- Preimage samples
- Output samples
Demo: Normal-Normal With Circular Condition

For $\varepsilon = 0.01$:

- Works fine with much smaller $\varepsilon$
Demo: Thermometer

• Normal-Normal thermometer process:

\[ X \sim \text{Normal}(90, 10) \]
\[ Y \sim \text{Normal}(X, 1) \]
\[ Y' = \min(100, Y) \]
Demo: Thermometer

• Normal-Normal thermometer process:

\[ X \sim \text{Normal}(90, 10) \]
\[ Y \sim \text{Normal}(X, 1) \]
\[ Y' = \min(100, Y) \]

• Objective: Find the distribution of \( X \mid Y' = 100 \)
Demo: Thermometer

• Normal-Normal thermometer process:

\[
X \sim \text{Normal}(90, 10) \\
Y \sim \text{Normal}(X, 1) \\
Y' = \min(100, Y)
\]

• Objective: Find the distribution of \( X \mid Y' = 100 \)

• Implementation:

\[
\text{(define/drbayes e)} \\
\quad \text{(let* ([x (normal 90 10)] [y (normal x 1)]) (list x (if (> y 100) 100 y))})
\]
Demo: Thermometer

• Normal-Normal thermometer process:

\[ X \sim \text{Normal}(90, 10) \]
\[ Y \sim \text{Normal}(X, 1) \]
\[ Y' = \min(100, Y) \]

• Objective: Find the distribution of \( X \mid Y' = 100 \)

• Implementation:

```
(define/drbayes e
  (let* ([x (normal 90 10)]
         [y (normal x 1)])
    (list x (if (> y 100) 100 y))))
```

• Goal: Sample in the preimage of

```
(set-list reals (interval 100 100))
```
Demo: Thermometer

Preimage rectangles
Demo: Thermometer

Preimage samples
Demo: Thermometer

Preimage samples

Density of $X \mid Y' = 100$
Demo: Thermometer

Preimage samples

Density of $X \mid Y' = 100$

Calculated from samples: mean 105.1, stddev 4.6
Demo: Stochastic Ray Tracing

• Idea: Model light transmission and reflection, condition on paths that pass through aperture
Demo: Stochastic Ray Tracing

- Idea: Model light transmission and reflection, condition on paths that pass through aperture
Demo: Stochastic Ray Tracing

• Part of the implementation (totals ~50 lines):

```scheme
(define/drbayes (ray-plane-intersect p0 v n d)
 (let ([denom (- (vec-dot v n))])
   (if (positive? denom)
     (let ([t (/ (+ d (vec-dot p0 n)) denom)])
       (if (positive? t)
         (collision t (vec+ p0 (vec-scale v t)) n)
         #f))
     #f)))
```
Demo: Stochastic Ray Tracing

• Part of the implementation (totals ~50 lines):

```lisp
(define/drbayes (ray-plane-intersect p0 v n d)
  (let [[denom (- (vec-dot v n))]]
    (if (positive? denom)
      (let ([t (/ (+ d (vec-dot p0 n)) denom)])
        (if (positive? t)
          (collision t (vec+ p0 (vec-scale v t)) n)
          #f))
      #f)))
```

• Constrained light path outputs:

![Paths Through Aperture](image1.png)
![Projected and Accumulated](image2.png)
Other Inference Tasks

• Typical
  ○ Hierarchical models
  ○ Bayesian regression
  ○ Model selection
Other Inference Tasks

• Typical
  ○ Hierarchical models
  ○ Bayesian regression
  ○ Model selection

• Atypical
  ○ Programs that halt with probability $\leq 1$, or never halt
  ○ Probabilistic program verification (sample in preimage of error condition)
Thesis Statement

Functional programming theory and measure-theoretic probability provide a solid foundation for trustworthy, useful languages for constructive probabilistic modeling and inference.
Functional programming theory and measure-theoretic probability provide a solid foundation for trustworthy, useful languages for constructive probabilistic modeling and inference.

---

True.
Thesis Statement

Functional programming theory and measure-theoretic probability provide a solid foundation for trustworthy, useful languages for constructive probabilistic modeling and inference.

True.

- Was it falsifiable?
Measurability

• Only *measurable* sets can have probabilities
Measurability

- Only *measurable* sets can have probabilities
- Computing preimages under $f$ must preserve measurability—we say $f$ itself is *measurable*
Measurability

• Only *measurable* sets can have probabilities

• Computing preimages under \( f \) must preserve measurability—we say \( f \) itself is *measurable*

---

**Theorem (measurability).** For all programs \( p \), \([p]\) is measurable, regardless of errors or nontermination, if language primitives are measurable.
Measurability

• Only *measurable* sets can have probabilities

• Computing preimages under \( f \) must preserve measurability—we say \( f \) itself is *measurable*

Theorem (measurability). For all programs \( p \), \([p]\) is measurable, regardless of errors or nontermination, if language primitives are measurable.

• Primitives include uncomputable operations like limits
Measurability

- Only *measurable* sets can have probabilities
- Computing preimages under $f$ must preserve measurability—we say $f$ itself is *measurable*

---

**Theorem (measurability).** For all programs $p$, $[p]$ is measurable, regardless of errors or nontermination, if language primitives are measurable.

---

- Primitives include uncomputable operations like limits
- Applies to all probabilistic programming languages
What I Did

Chapter 1
Thesis

Chapter 2
Background

Chapter 3
Related Work

Chapter 4
$\lambda_{\text{ZFC}}$

Chapter 5
Using $\lambda_{\text{ZFC}}$

Chapter 6
Countable Models and Implementation

Chapter 7
Interlude

Chapter 8
Preimage Computation Theory

Appendix A
Measurability Theorems

Chapter 9
Preimage Computation Implementation

Appendix B
Sampling Theorems

Chapter 10
Example Programs
What I Did

The core calculus for this:
Future Work

• Expressiveness
  ◦ Lambdas and macros
  ◦ Exceptions, parameters (or continuations and marks)
Future Work

• Expressiveness
  ◦ Lambdas and macros
  ◦ Exceptions, parameters (or continuations and marks)

• Optimization
  ◦ Direct implementation is $O(n^2)$ in depth; cut to $O(n)$
  ◦ Incremental computation
  ◦ Adaptive sampling algorithms
  ◦ Static analysis
Future Work

- Expressiveness
  - Lambdas and macros
  - Exceptions, parameters (or continuations and marks)

- Optimization
  - Direct implementation is $O(n^2)$ in depth; cut to $O(n)$
  - Incremental computation
  - Adaptive sampling algorithms
  - Static analysis

- Branching out: investigate preimage computation connection with type systems and predicate transformer semantics