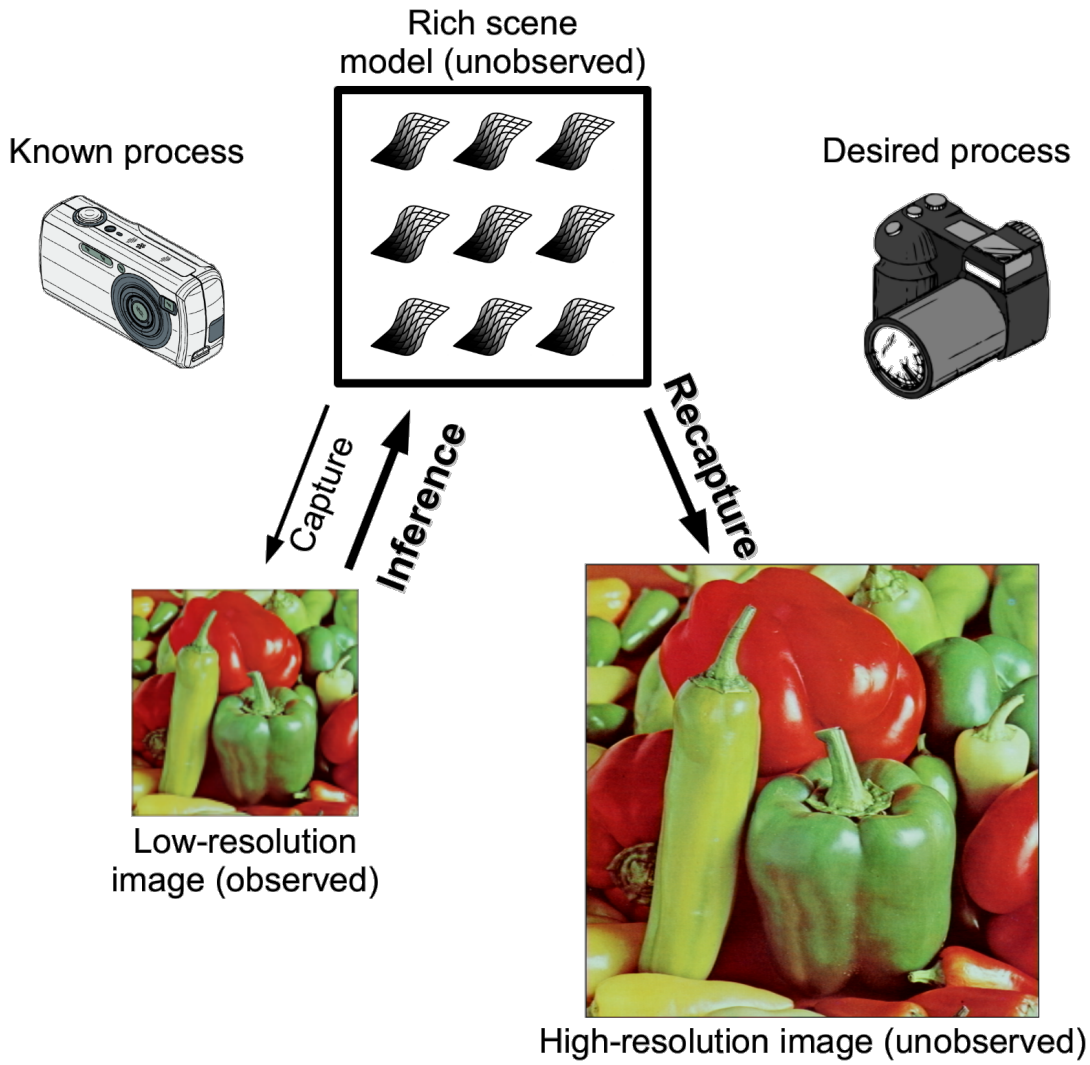
Running Probabilistic Programs Backward

Neil Toronto

PLT @ Brigham Young University

Master's Research: Super-Resolution



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

$$\mathbf{C}^x_{i,j} \equiv i + \tfrac{1}{2} \qquad i \in 0..m-1$$
$$\mathbf{C}^y_{i,j} \equiv j + \tfrac{1}{2} \qquad j \in 0..n-1$$

$$\mathrm{N9}(x,y) \equiv \{i \in \mathbb{Z} \mid -1 \le i - \lfloor x \rfloor \le 1\} \times \{j \in \mathbb{Z} \mid -1 \le j - \lfloor y \rfloor \le 1\}$$

$$\mathrm{dist}(x,y,\theta,d) \equiv x\cos\theta + y\sin\theta - d$$

$$\mathrm{prof}(d,\sigma,v^+,v^-) \equiv \tfrac{v^+ - v^-}{2}\,\mathrm{erf}\left(\tfrac{d}{\sqrt{2}\sigma}\right) + \tfrac{v^+ + v^-}{2}$$

$$\mathrm{edge}(x,y,\theta,d,v^+,v^-,\sigma) \equiv \mathrm{prof}(\mathrm{dist}(x,y,\theta,d),\sigma,v^+,v^-)$$

$$\mathbf{S}^{\mathrm{edge}}_{i,j}(x,y) \equiv \mathrm{edge}(x - \mathbf{C}^x_{i,j}, y - \mathbf{C}^y_{i,j}, \mathbf{S}^\theta_{i,j}, \mathbf{S}^d_{i,j}, \mathbf{S}^{v^+}_{i,j}, \mathbf{S}^{v^-}_{i,j}, \mathbf{S}^\sigma_{i,j})$$

$$\mathrm{E}[h(\mathbf{S}_{x,y})] \equiv \sum_{k,l \in \mathrm{N9}(x,y)} w(x - \mathbf{C}^x_{k,l}, y - \mathbf{C}^y_{k,l})\; h(\mathbf{S}^{\mathrm{edge}}_{k,l}(x,y))$$

$$\mathbf{S}^\theta_{i,j} \sim \mathrm{Uniform}(-\pi,\pi) \qquad \mathbf{S}^{v^+}_{i,j} \sim \mathrm{Uniform}(0,1)$$
$$\mathbf{S}^d_{i,j} \sim \mathrm{Uniform}(-3,3) \qquad \mathbf{S}^{v^-}_{i,j} \sim \mathrm{Uniform}(0,1)$$
$$\mathbf{S}^\sigma_{i,j} \sim \mathrm{Beta}(1.6,1)$$

$$\mathbf{I}_{i,j} | \mathbf{S}_{\mathrm{N9}(i,j)} \sim \mathrm{Normal}(\mathrm{E}[\mathbf{S}_{i,j}], \omega)$$

$$\Phi_{i,j}(\mathbf{S}_{\mathrm{N9}(i,j)}) \equiv \exp\left(-\frac{\mathrm{Var}[\mathbf{S}_{i,j}]}{2\gamma^2}\right)$$

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:

Resolution Synthesis

Bayesian Edge Inference

Main Results: Super-Resolution

- Competitor and BEI on 4x super-resolution:

Resolution Synthesis

Bayesian Edge Inference

- Also beat state-of-the-art on “objective” measures

Pleasantly Surprising Results: Inpainting

- Advisor: “Hey, doesn’t Bayesian inference handle missing data easily?”

Pleasantly Surprising Results: Inpainting

- Advisor: “Hey, doesn’t Bayesian inference handle missing data easily?”
- Me: Yep. Gimme 20 minutes...

Pleasantly Surprising Results: Inpainting

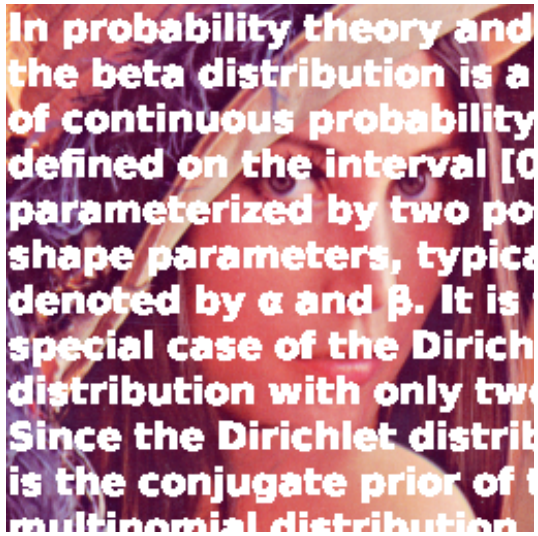
- Advisor: “Hey, doesn’t Bayesian inference handle missing data easily?”
- Me: Yep. Gimme 20 minutes...

Original Image

Pleasantly Surprising Results: Inpainting

- Advisor: “Hey, doesn’t Bayesian inference handle missing data easily?”
- Me: Yep. Gimme 20 minutes...

Original Image

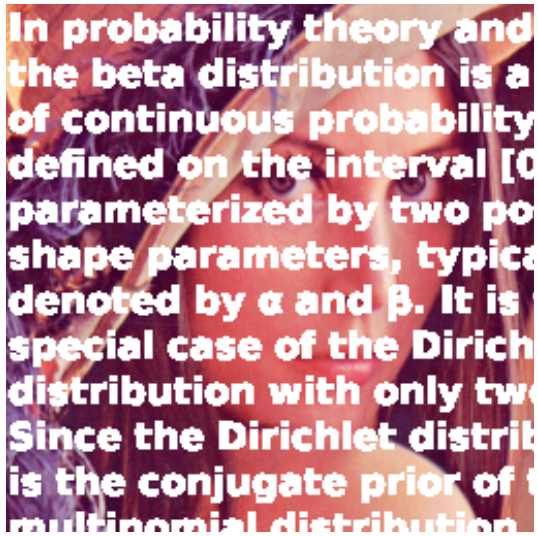


33% Defaced

Pleasantly Surprising Results: Inpainting

- Advisor: “Hey, doesn’t Bayesian inference handle missing data easily?”
- Me: Yep. Gimme 20 minutes...

Original Image



33% Defaced

Bayesian Edge Inference

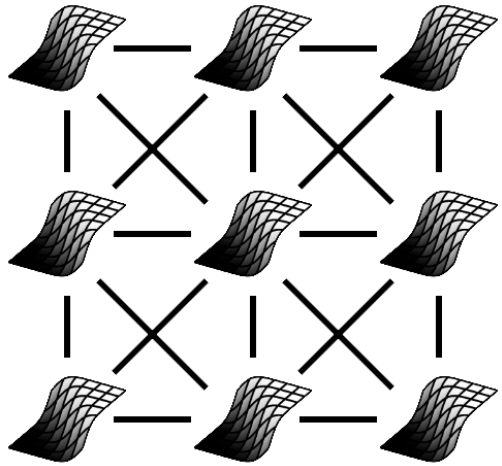
Only Mostly Satisfying

Reason 1: Still not sure the program is right

Only Mostly Satisfying

Reason 1: Still not sure the program is right

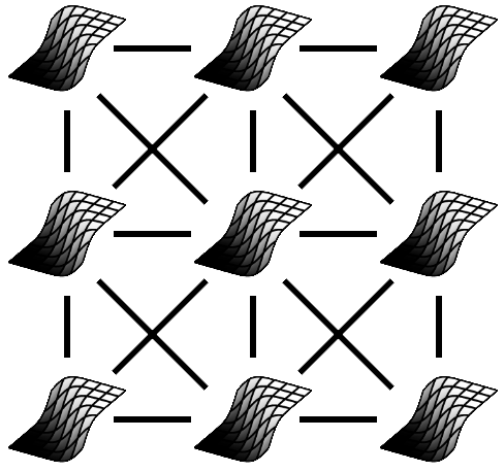
Scene: overlapping, smooth, local edges



Only Mostly Satisfying

Reason 1: Still not sure the program is right

Scene: overlapping, smooth, local edges

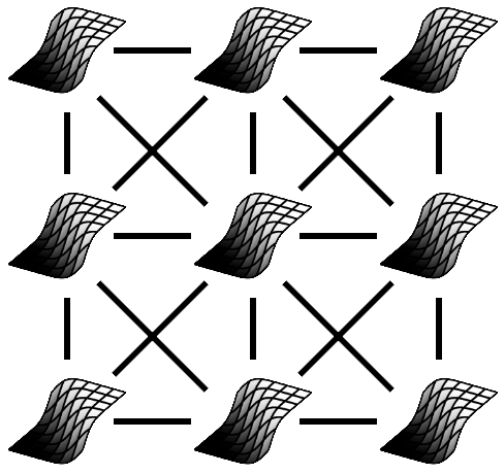


Reason 2: “To *approximate* blurring with a spatially varying point-spread function (PSF), we assign each facet a Gaussian PSF and convolve each analytically *before combining outputs*.”

Only Mostly Satisfying

Reason 1: Still not sure the program is right

Scene: overlapping, smooth, local edges



Reason 2: “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.”

Simple Example Process

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

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

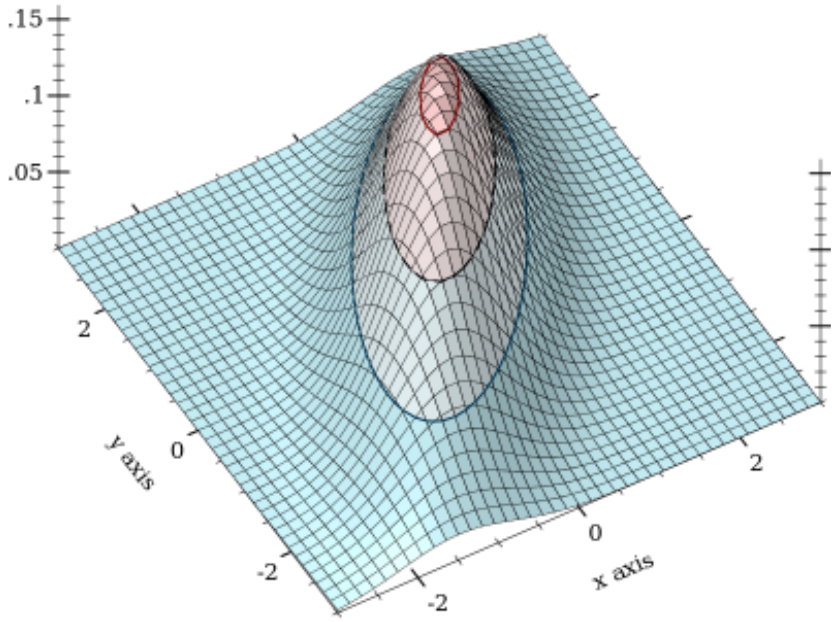
- Intuition: sample X, then sample Y using X

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 $f : \mathbb{R} \times \mathbb{R} \to [0, \infty)$:



Observation Queries

Observation Queries

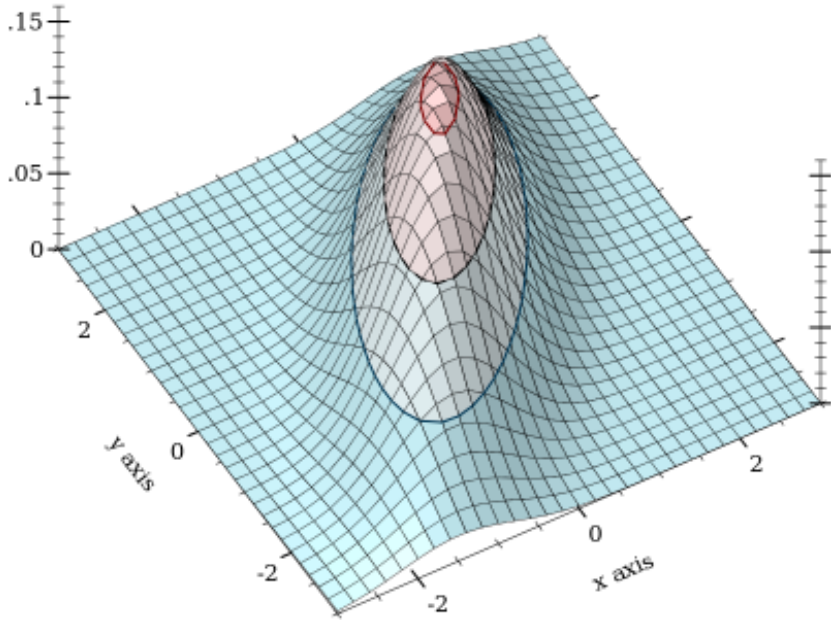
- What's the distribution of $X \mid Y = 2$? Use Bayes' law:

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

Observation Queries

- What's the distribution of $X \mid Y = 2$? Use Bayes' law:

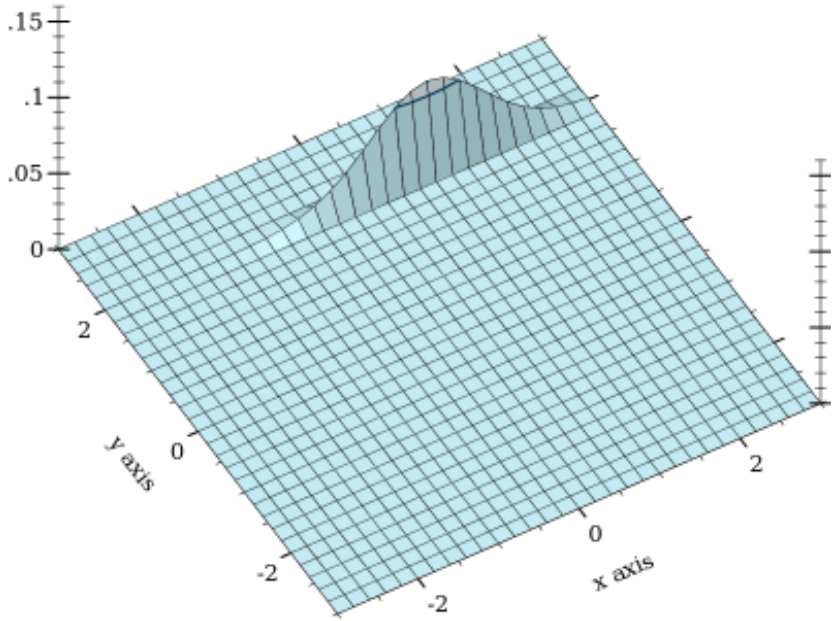
$$f(x \mid y) = \frac{f(x, y)}{\int_{-\infty}^{\infty} f(x, y)\ dx}$$



Observation Queries

- What's the distribution of $X \mid Y = 2$? Use Bayes' law:

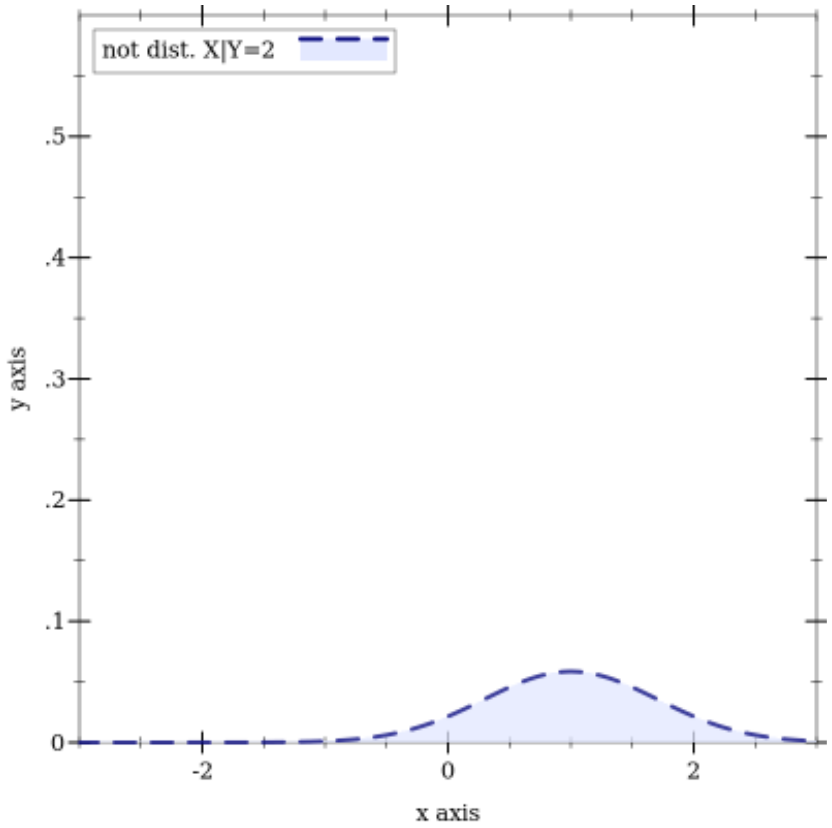
$$f(x \mid y) = \frac{f(x, y)}{\int_{-\infty}^{\infty} f(x, y)\ dx}$$



Observation Queries

- What's the distribution of $X \mid Y = 2$? Use Bayes' law:

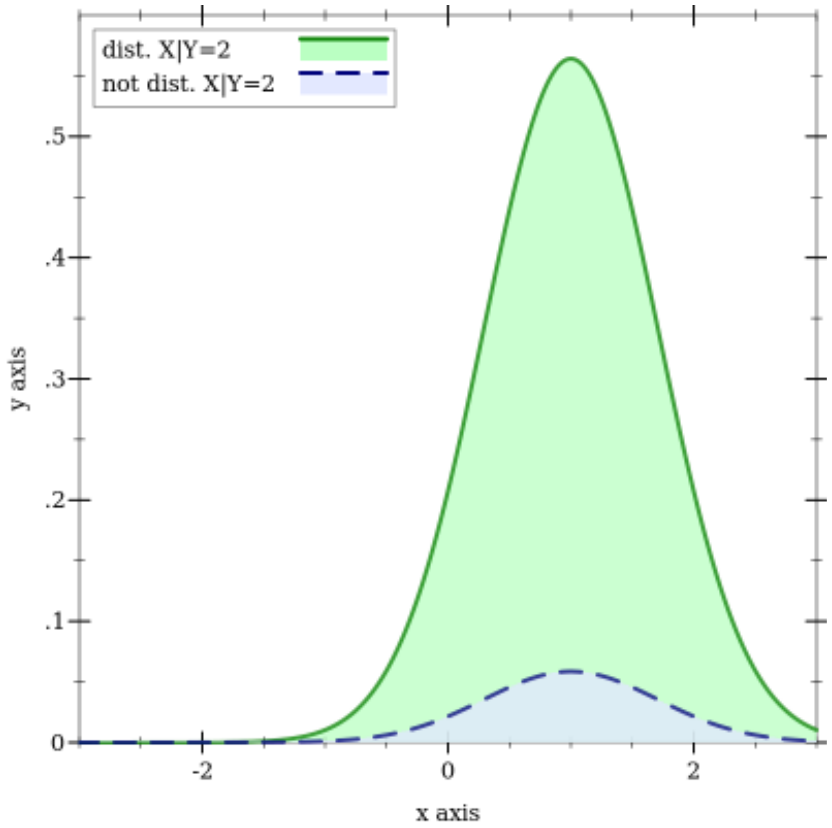
$$f(x \mid y) = \frac{f(x, y)}{\int_{-\infty}^{\infty} f(x, y)\ dx}$$



Observation Queries

- What's the distribution of $X \mid Y = 2$? Use Bayes' law:

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



An Observation Puzzle

- Find the distribution of $X, Y \mid \sqrt{X^2 + Y^2} = 1$

An Observation Puzzle

- Find the distribution of $X, Y \mid \sqrt{X^2 + Y^2} = 1$
- Can’t reduce dimensions by chopping one off

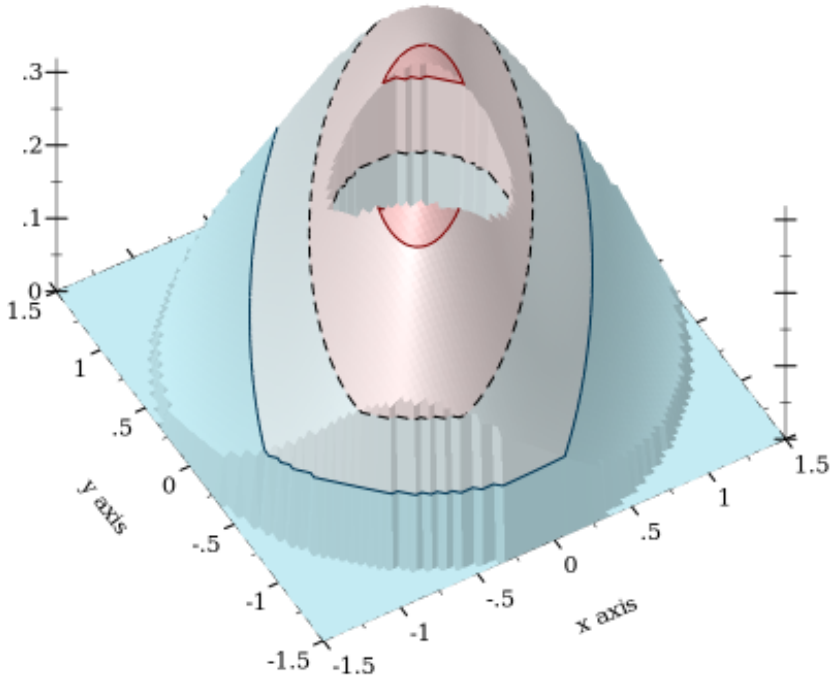
An Observation Puzzle

- Find the distribution of $X, Y \mid \sqrt{X^2 + Y^2} = 1$
- Can't reduce dimensions by chopping one off
- Maybe take a limit of $\left|\sqrt{X^2 + Y^2} - 1\right| < \epsilon$?

An Observation Puzzle

- Find the distribution of $X, Y \mid \sqrt{X^2 + Y^2} = 1$
- Can't reduce dimensions by chopping one off
- Maybe take a limit of $\left|\sqrt{X^2 + Y^2} - 1\right| < \epsilon$?

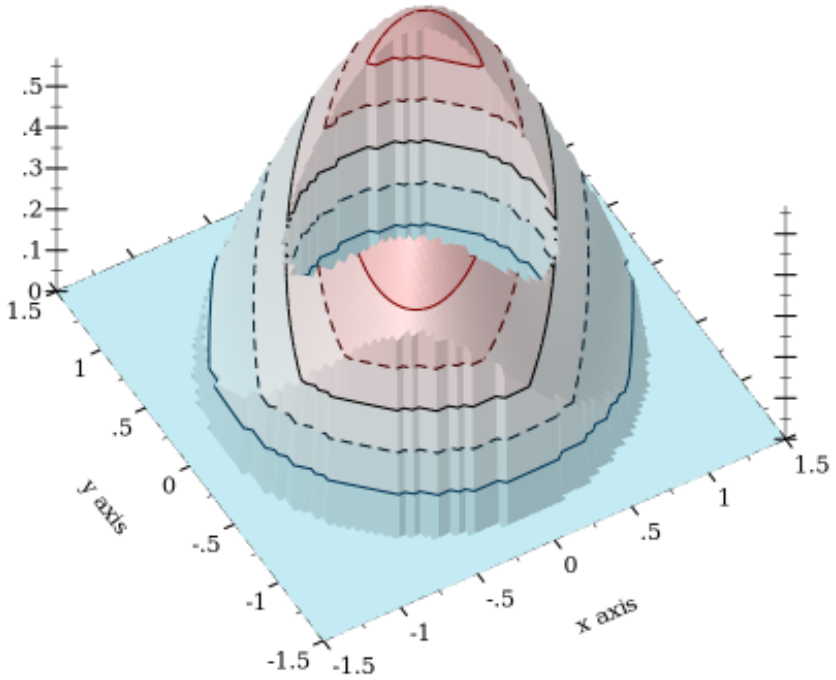
Conditional density with $\epsilon = 0.5$:



An Observation Puzzle

- Find the distribution of $X, Y \mid \sqrt{X^2 + Y^2} = 1$
- Can't reduce dimensions by chopping one off
- Maybe take a limit of $\left|\sqrt{X^2 + Y^2} - 1\right| < \epsilon$?

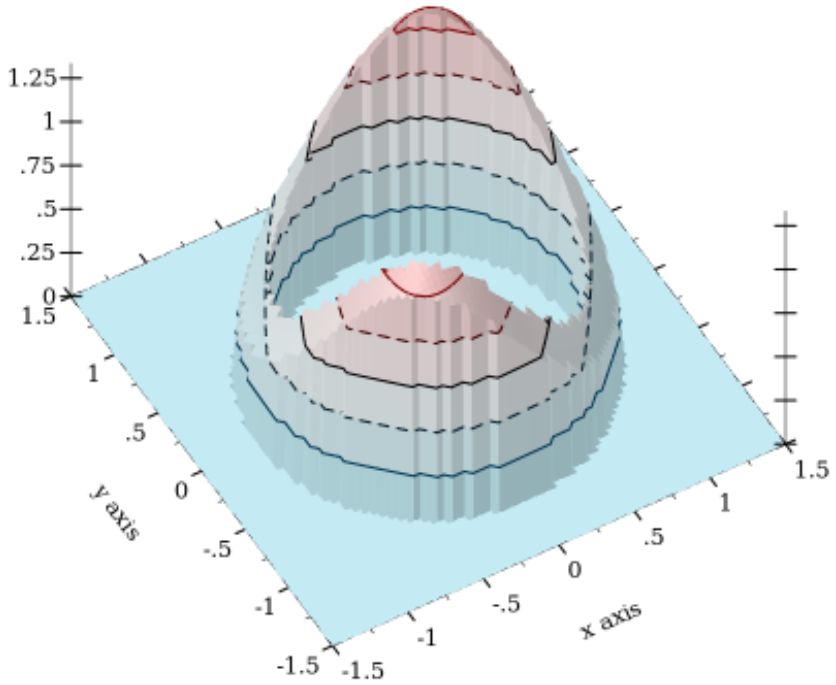
Conditional density with $\epsilon = 0.25$:



An Observation Puzzle

- Find the distribution of $X, Y \mid \sqrt{X^2 + Y^2} = 1$
- Can't reduce dimensions by chopping one off
- Maybe take a limit of $\left|\sqrt{X^2 + Y^2} - 1\right| < \epsilon$?

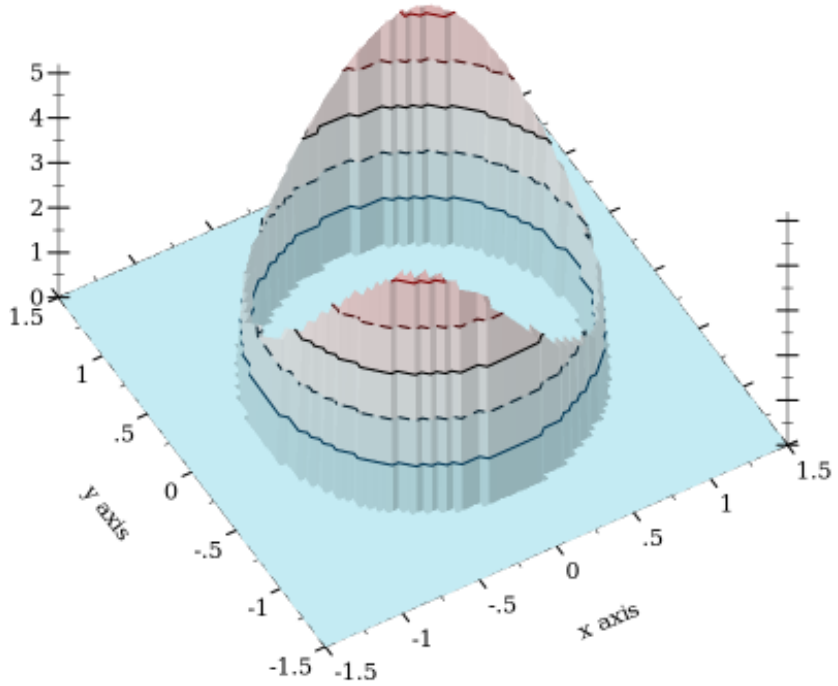
Conditional density with $\epsilon = 0.1$:



An Observation Puzzle

- Find the distribution of $X, Y \mid \sqrt{X^2 + Y^2} = 1$
- Can't reduce dimensions by chopping one off
- Maybe take a limit of $\left|\sqrt{X^2 + Y^2} - 1\right| < \epsilon$?

Conditional density with $\epsilon = 0.025$:



What Can't Densities Model?

- Non-axis-aligned, zero-probability conditions (can't reduce dimension)

What Can't Densities Model?

- Non-axis-aligned, zero-probability conditions (can't reduce dimension)
- CDFs with discontinuities

What Can't Densities Model?

- Non-axis-aligned, zero-probability conditions (can't reduce dimension)
- CDFs with discontinuities
- Discontinuous change of variable (e.g. a thermometer)

What Can't Densities Model?

- Non-axis-aligned, zero-probability conditions (can't reduce dimension)
- CDFs with discontinuities
- Discontinuous change of variable (e.g. a thermometer)
- Distributions with variable-dimension support

What Can't Densities Model?

- Non-axis-aligned, zero-probability conditions (can't reduce dimension)
- CDFs with discontinuities
- Discontinuous change of variable (e.g. a thermometer)
- Distributions with variable-dimension support
- Nontrivial distributions on infinite products like $[0, 1]^{\mathbb{N}}$

What Can't Densities Model?

- Non-axis-aligned, zero-probability conditions (can't reduce dimension)
- CDFs with discontinuities
- Discontinuous change of variable (e.g. a thermometer)
- Distributions with variable-dimension support
- Nontrivial distributions on infinite products like $[0, 1]^{\mathbb{N}}$

There are tricks to get around limitations, but none are generally applicable...

Measure-Theoretic Probability

- Main ideas:
  - Don’t assign *changes* in probability to *values*, assign *probabilities* to *sets*

Measure-Theoretic Probability

- Main ideas:
  - Don’t assign *changes* in probability to *values*, assign *probabilities* to *sets*
  - 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 *changes* in probability to *values*, assign *probabilities* to *sets*
  - Confine assumed randomness to one place by making random variables *deterministic functions* that observe a random source
- Measure-theoretic model of example process:

$$\Omega = \mathbb{R} \times \mathbb{R}$$

$$P : \mathrm{Set}(\Omega) \rightharpoonup [0, 1], \;\; P(A) = \int_A f \; d\lambda$$

Measure-Theoretic Probability

- Main ideas:
  - Don't assign *changes* in probability to *values*, assign *probabilities* to *sets*
  - Confine assumed randomness to one place by making random variables *deterministic functions* that observe a random source
- Measure-theoretic model of example process:

$$\Omega = \mathbb{R} \times \mathbb{R}$$
$$P : \mathrm{Set}(\Omega) \rightharpoonup [0, 1], \;\; P(A) = \int_A f \; d\lambda$$
$$X : \Omega \to \mathbb{R}, \;\; X(\omega) = \omega_0$$
$$Y : \Omega \to \mathbb{R}, \;\; Y(\omega) = \omega_1$$

Measure-Theoretic Queries

- Specific query:

$$\Pr[X > 1] = P(\{\omega \in \Omega \mid X(\omega) > 1\})$$

Measure-Theoretic Queries

- Specific query:

$$\Pr[X > 1] = P(\{\omega \in \Omega \mid X(\omega) > 1\})$$

- Generalized:

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

Measure-Theoretic Queries

- Specific query:

$$\Pr[X > 1] = P(\{\omega \in \Omega \mid X(\omega) > 1\})$$

- Generalized:

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

- Conditional query:

$$\Pr[e_1 \mid e_2] = \Pr[e_1, e_2]/\Pr[e_2] \quad \text{if } \Pr[e_2] > 0$$

Measure-Theoretic Queries

- Specific query:

$$\Pr[X > 1] = P(\{\omega \in \Omega \mid X(\omega) > 1\})$$

- Generalized:

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

- Conditional query:

$$\Pr[e_1 \mid e_2] = \Pr[e_1, e_2]/\Pr[e_2] \quad \text{if } \Pr[e_2] > 0$$

But $\Pr[Y = 2] = 0$...

Zero-Probability Conditions (Axial)

- Observation query:

$$\Pr[X > 1 \mid Y = 2] \;=\; \lim_{\epsilon \to 0} \Pr[X > 1 \mid |Y - 2| < \epsilon]$$

Zero-Probability Conditions (Axial)

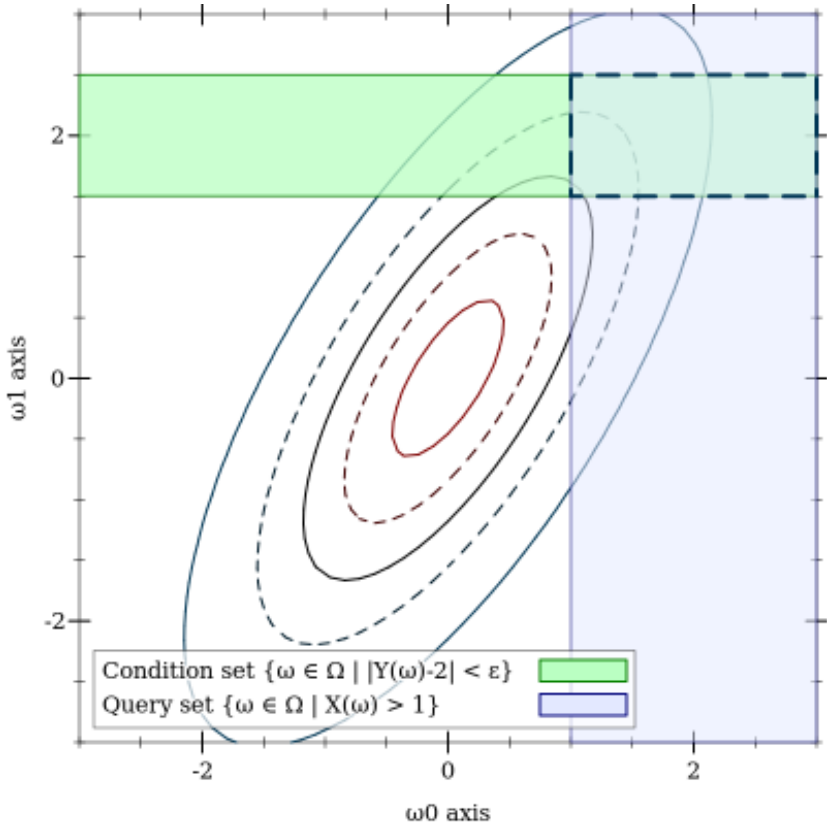
- Observation query:

$$\begin{aligned}\Pr[X > 1 \,|\, Y = 2] &= \lim_{\epsilon \to 0} \Pr[X > 1 \,|\, |Y - 2| < \epsilon] \\ &= \lim_{\epsilon \to 0} \frac{\Pr[X > 1, |Y - 2| < \epsilon]}{\Pr[|Y - 2| < \epsilon]}\end{aligned}$$

Zero-Probability Conditions (Axial)

- Observation query:

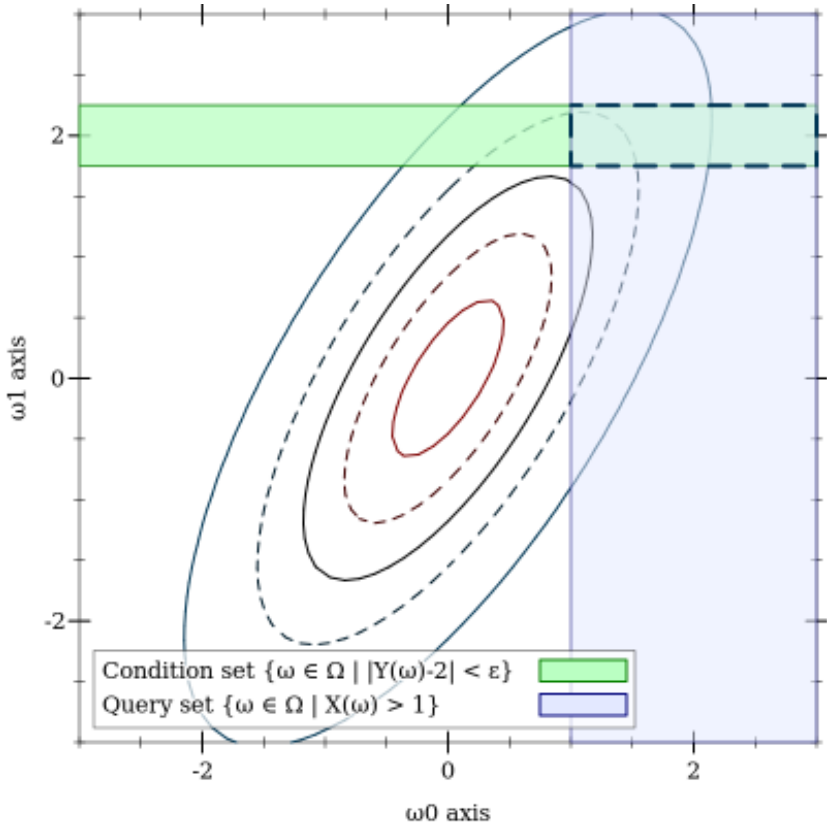
$$\begin{aligned}\Pr[X > 1 \mid Y = 2] &= \lim_{\epsilon \to 0} \Pr[X > 1 \mid |Y - 2| < \epsilon] \\ &= \lim_{\epsilon \to 0} \frac{\Pr[X > 1, |Y - 2| < \epsilon]}{\Pr[|Y - 2| < \epsilon]}\end{aligned}$$



Zero-Probability Conditions (Axial)

- Observation query:

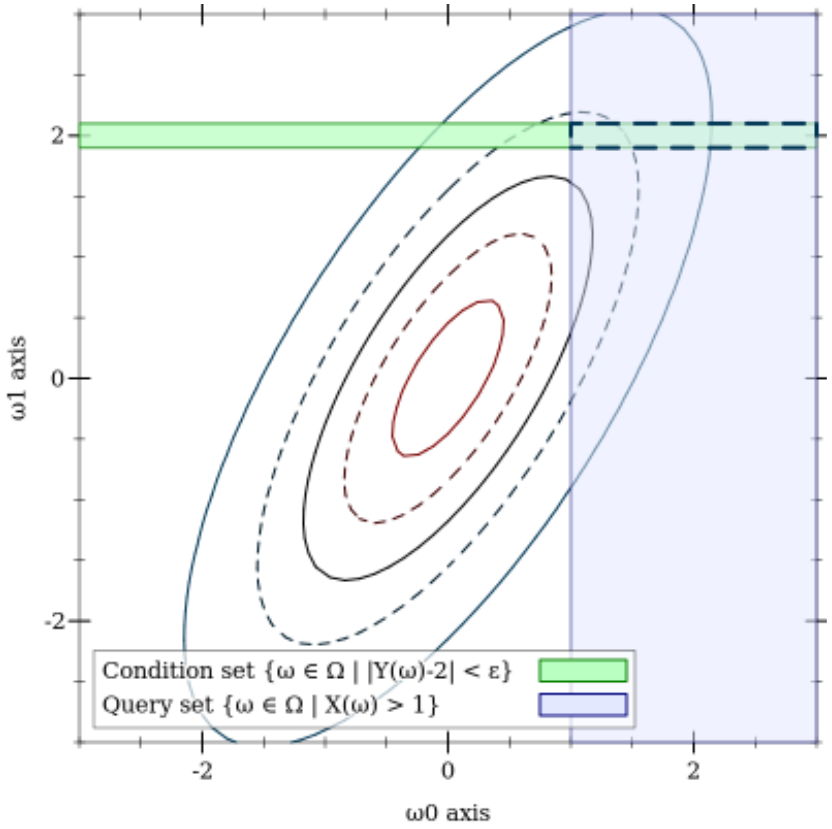
$$\begin{aligned}\Pr[X > 1 \mid Y = 2] &= \lim_{\epsilon \to 0} \Pr[X > 1 \mid |Y - 2| < \epsilon] \\ &= \lim_{\epsilon \to 0} \frac{\Pr[X > 1, |Y - 2| < \epsilon]}{\Pr[|Y - 2| < \epsilon]}\end{aligned}$$



Zero-Probability Conditions (Axial)

- Observation query:

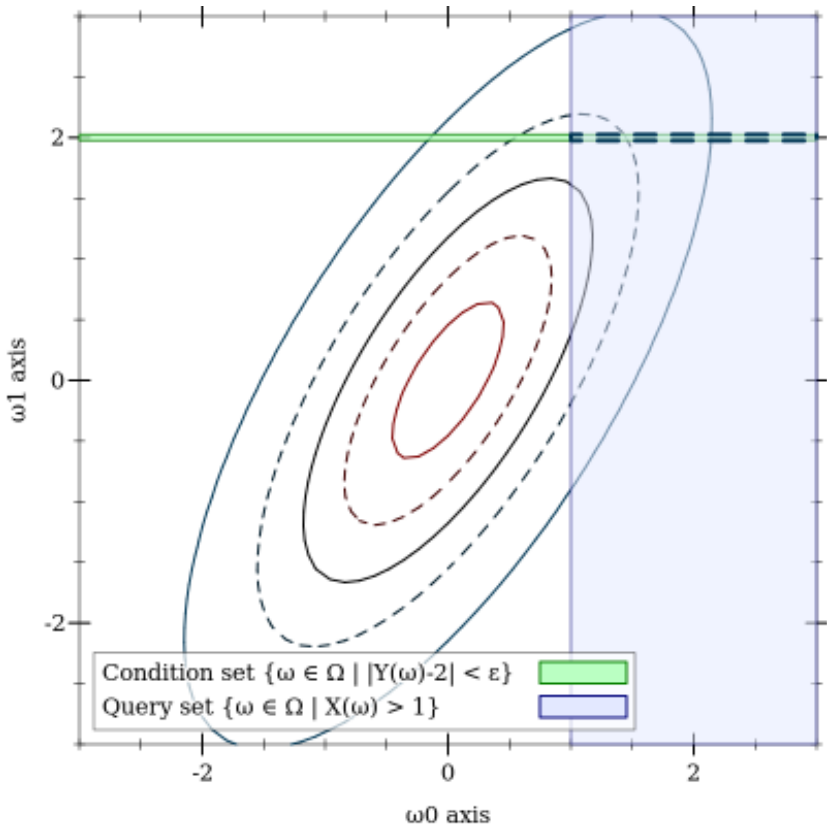
$$\begin{aligned}\Pr[X > 1 \mid Y = 2] &= \lim_{\epsilon \to 0} \Pr[X > 1 \mid |Y - 2| < \epsilon] \\ &= \lim_{\epsilon \to 0} \frac{\Pr[X > 1, |Y - 2| < \epsilon]}{\Pr[|Y - 2| < \epsilon]}\end{aligned}$$



Zero-Probability Conditions (Axial)

- Observation query:

$$\begin{aligned}\Pr[X > 1 \mid Y = 2] &= \lim_{\epsilon \to 0} \Pr[X > 1 \mid |Y - 2| < \epsilon] \\ &= \lim_{\epsilon \to 0} \frac{\Pr[X > 1, |Y - 2| < \epsilon]}{\Pr[|Y - 2| < \epsilon]}\end{aligned}$$



Zero-Probability Conditions (Circular)

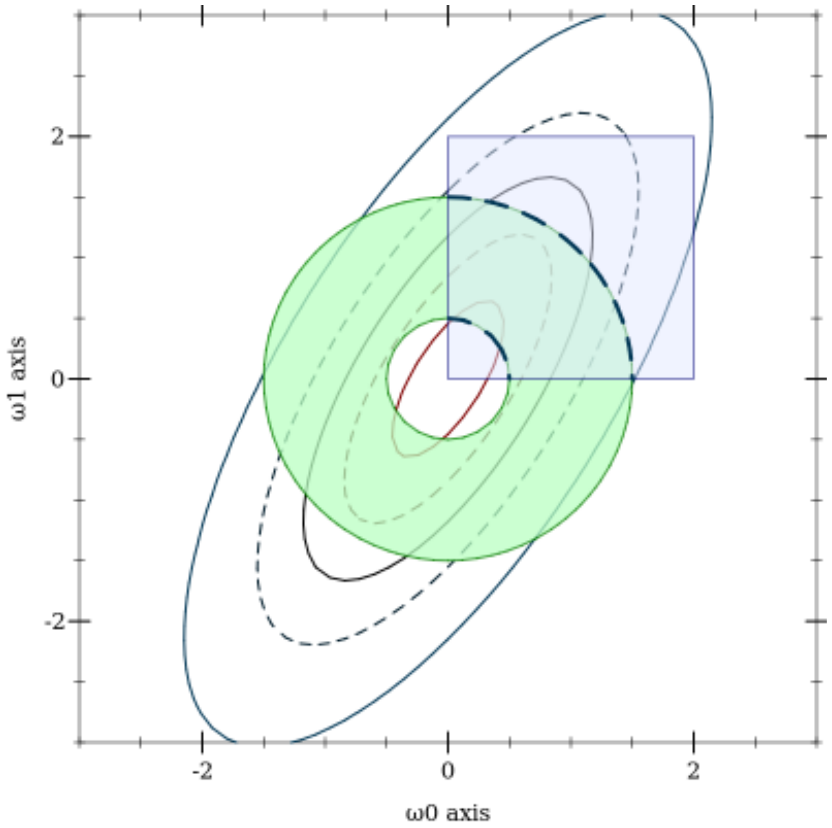
- Can condition w.r.t. any random variable:

$$\Pr[e \mid \sqrt{X^2 + Y^2} = 1] \;=\; \lim_{\epsilon \to 0} \Pr[e \mid |\sqrt{X^2 + Y^2} - 1| < \epsilon]$$

Zero-Probability Conditions (Circular)

- Can condition w.r.t. any random variable:

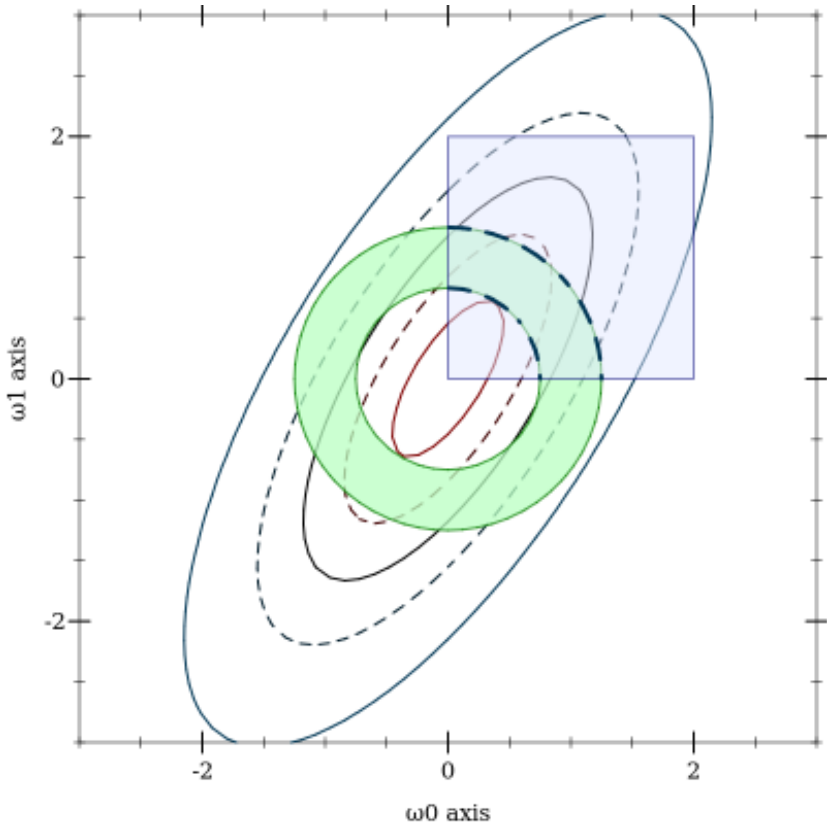
$$\Pr[e \mid \sqrt{X^2 + Y^2} = 1] \;=\; \lim_{\epsilon \to 0} \Pr[e \mid |\sqrt{X^2 + Y^2} - 1| < \epsilon]$$



Zero-Probability Conditions (Circular)

- Can condition w.r.t. any random variable:

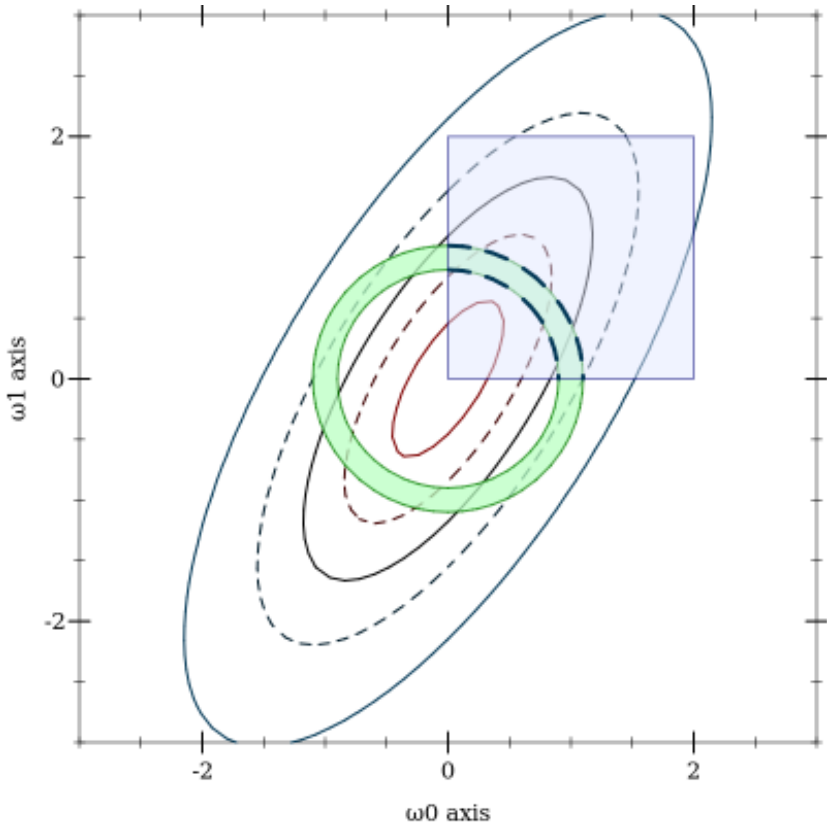
$$\Pr[e \mid \sqrt{X^2 + Y^2} = 1] \;=\; \lim_{\epsilon \to 0} \Pr[e \mid |\sqrt{X^2 + Y^2} - 1| < \epsilon]$$



Zero-Probability Conditions (Circular)

- Can condition w.r.t. any random variable:

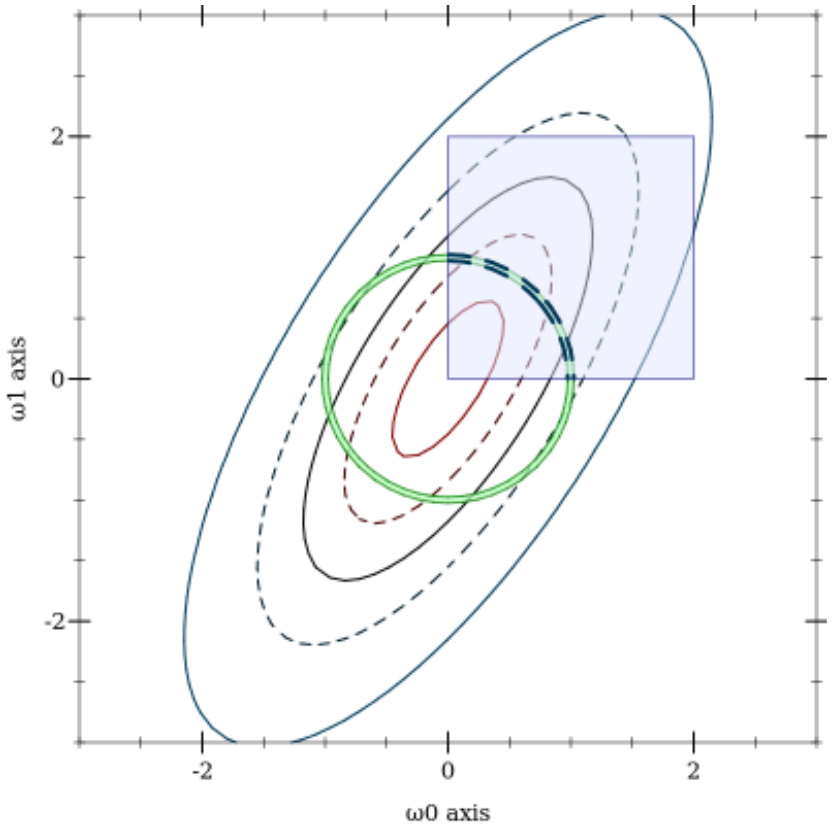
$$\Pr[e \mid \sqrt{X^2 + Y^2} = 1] \;=\; \lim_{\epsilon \to 0} \Pr[e \mid |\sqrt{X^2 + Y^2} - 1| < \epsilon]$$



Zero-Probability Conditions (Circular)

- Can condition w.r.t. any random variable:

$$\Pr[e \mid \sqrt{X^2 + Y^2} = 1] \;=\; \lim_{\epsilon \to 0} \Pr[e \mid |\sqrt{X^2 + Y^2} - 1| < \epsilon]$$



Measure-Theoretic Models

- Random variables and $\Pr[\cdot]$ are an abstraction boundary that hides Ω and P

Measure-Theoretic Models

- Random variables and $\Pr[\cdot]$ are an abstraction boundary that hides Ω and P
- Queries return the same answers for any model whose random variables' outputs have the correct joint distribution

Measure-Theoretic Models

- Random variables and $\Pr[\cdot]$ are an abstraction boundary that hides Ω and P
- Queries return the same answers for any model whose random variables' outputs have the correct joint distribution

Let $F : \mathbb{R} \to [0, 1]$ be the Normal CDF, and define a **uniform random source** model:

$$\Omega = [0, 1] \times [0, 1]$$
$$P : \mathrm{Set}(\Omega) \rightharpoonup [0, 1], \;\; P(A) = \lambda(A) \;\; (\text{i.e. } A\text{'s area})$$

Measure-Theoretic Models

- Random variables and $\Pr[\cdot]$ are an abstraction boundary that hides Ω and P
- Queries return the same answers for any model whose random variables' outputs have the correct joint distribution

Let $F : \mathbb{R} \to [0, 1]$ be the Normal CDF, and define a **uniform random source** model:

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

Now We Finally Get to Go Backward

- Generalized query:

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

Now We Finally Get to Go Backward

- Generalized query:

$$\begin{aligned}\Pr[Z \in C] &= P(\{\omega \in \Omega \mid Z(\omega) \in C\}) \\ &= P(Z^{-1}(C))\end{aligned}$$

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

Now We Finally Get to Go Backward

- Generalized query:

$$\begin{aligned}\Pr[Z \in C] &= P(\{\omega \in \Omega \mid Z(\omega) \in C\}) \\ &= P(Z^{-1}(C))\end{aligned}$$

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

- For a uniform random source model,
  - Compute output probabilities by computing **preimage** areas

Now We Finally Get to Go Backward

- Generalized query:

$$\begin{aligned}\Pr[Z \in C] &= P(\{\omega \in \Omega \mid Z(\omega) \in C\}) \\ &= P(Z^{-1}(C))\end{aligned}$$

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

- For a uniform random source model,
  - Compute output probabilities by computing **preimage** areas
  - Compute conditional probabilities as quotients of **preimage** areas

Now We Finally Get to Go Backward

- Generalized query:

$$\begin{aligned}\Pr[Z \in C] &= P(\{\omega \in \Omega \mid Z(\omega) \in C\}) \\ &= P(Z^{-1}(C))\end{aligned}$$

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

- For a uniform random source model,
  - Compute output probabilities by computing **preimage** areas
  - Compute conditional probabilities as quotients of **preimage** areas
- What do preimages look like?

Preimages Under the Unit Circle Condition

- Random variable for the unit circle condition:

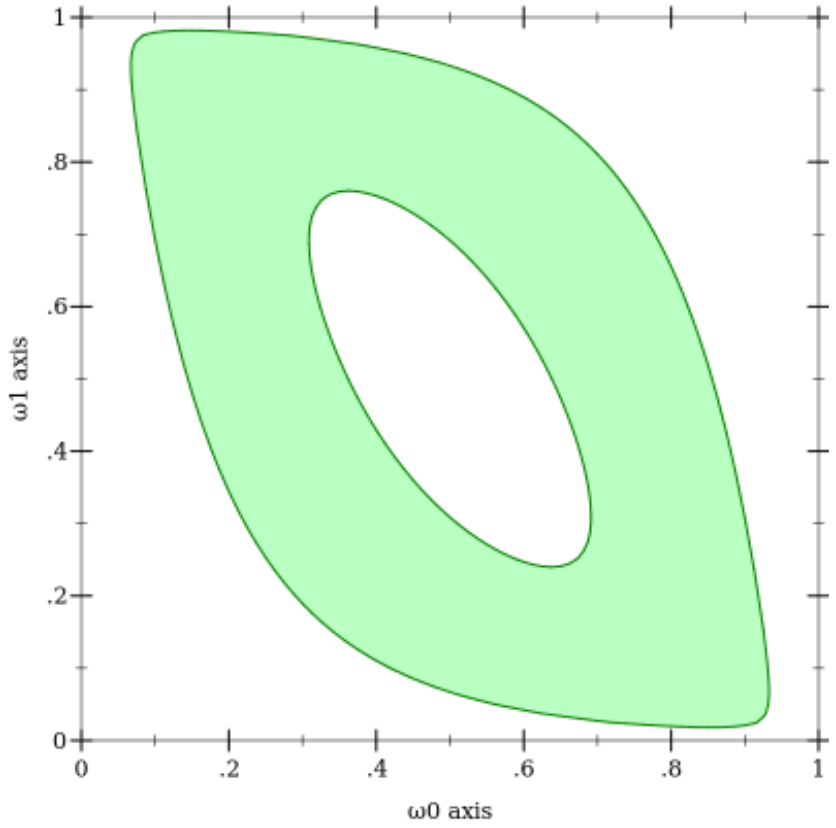
$$Z(\omega) = |\sqrt{X(\omega)^2 + Y(\omega)^2} - 1|$$

Preimages Under the Unit Circle Condition

- Random variable for the unit circle condition:

$$Z(\omega) = |\sqrt{X(\omega)^2 + Y(\omega)^2} - 1|$$

$Z^{-1}([0, \epsilon)), \epsilon = 0.5$:

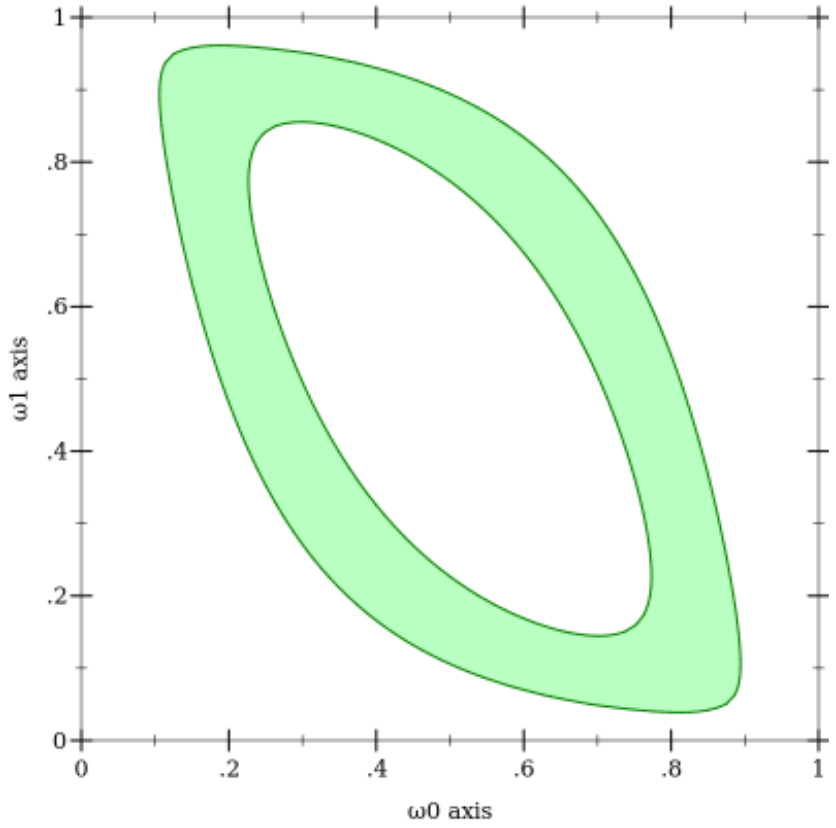


Preimages Under the Unit Circle Condition

- Random variable for the unit circle condition:

$$Z(\omega) = |\sqrt{X(\omega)^2 + Y(\omega)^2} - 1|$$

$Z^{-1}([0, \epsilon)), \epsilon = 0.25$:

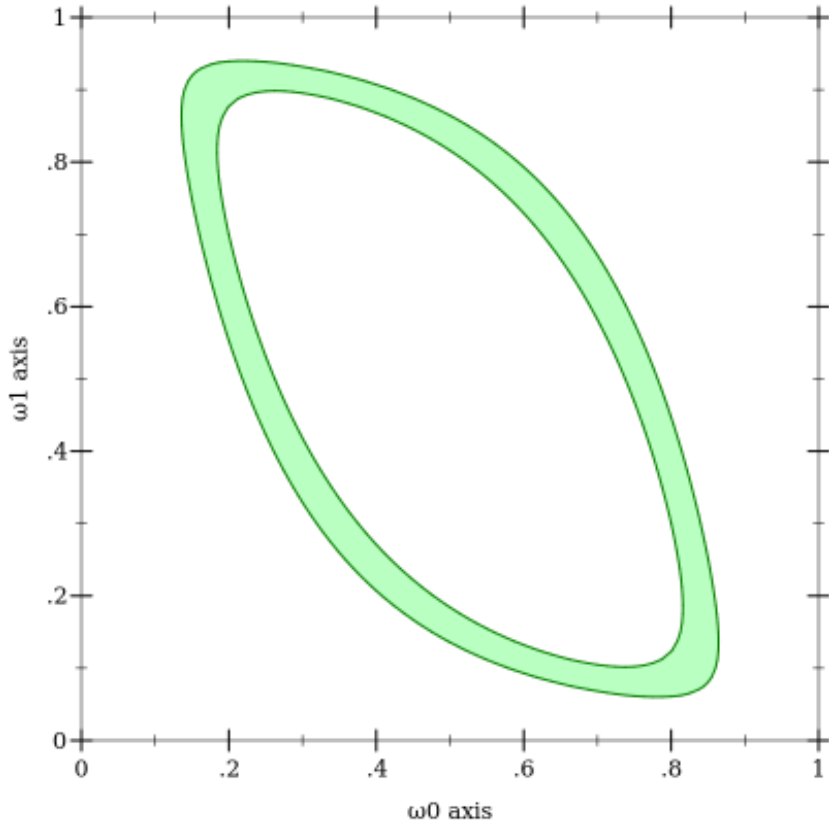


Preimages Under the Unit Circle Condition

- Random variable for the unit circle condition:

$$Z(\omega) = |\sqrt{X(\omega)^2 + Y(\omega)^2} - 1|$$

$Z^{-1}([0, \epsilon)), \epsilon = 0.1$:

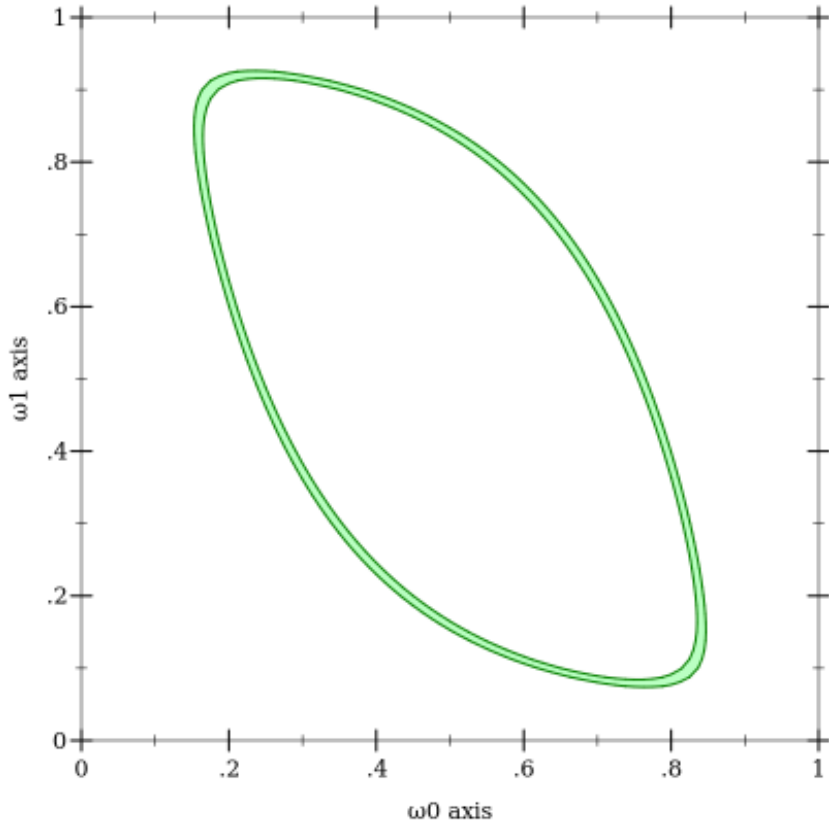


Preimages Under the Unit Circle Condition

- Random variable for the unit circle condition:

$$Z(\omega) = |\sqrt{X(\omega)^2 + Y(\omega)^2} - 1|$$

$Z^{-1}([0, \epsilon)), \epsilon = 0.025$:



Crazy Idea

- Define a language for probabilistic reasoning using measure theory

Crazy Idea

- Define a language for probabilistic reasoning using measure theory
- Language should:
  - Allow arbitrary probabilistic conditions (positive-probability for now)
  - Allow recursion and unbounded loops

Crazy Idea

- Define a language for probabilistic reasoning using measure theory
- Language should:
  - Allow arbitrary probabilistic conditions (positive-probability for now)
  - Allow recursion and unbounded loops
- Implementation should:
  - Answer probability queries efficiently
  - Operate correctly

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 volumes of preimage sets

Crazy Idea is Feasible If...

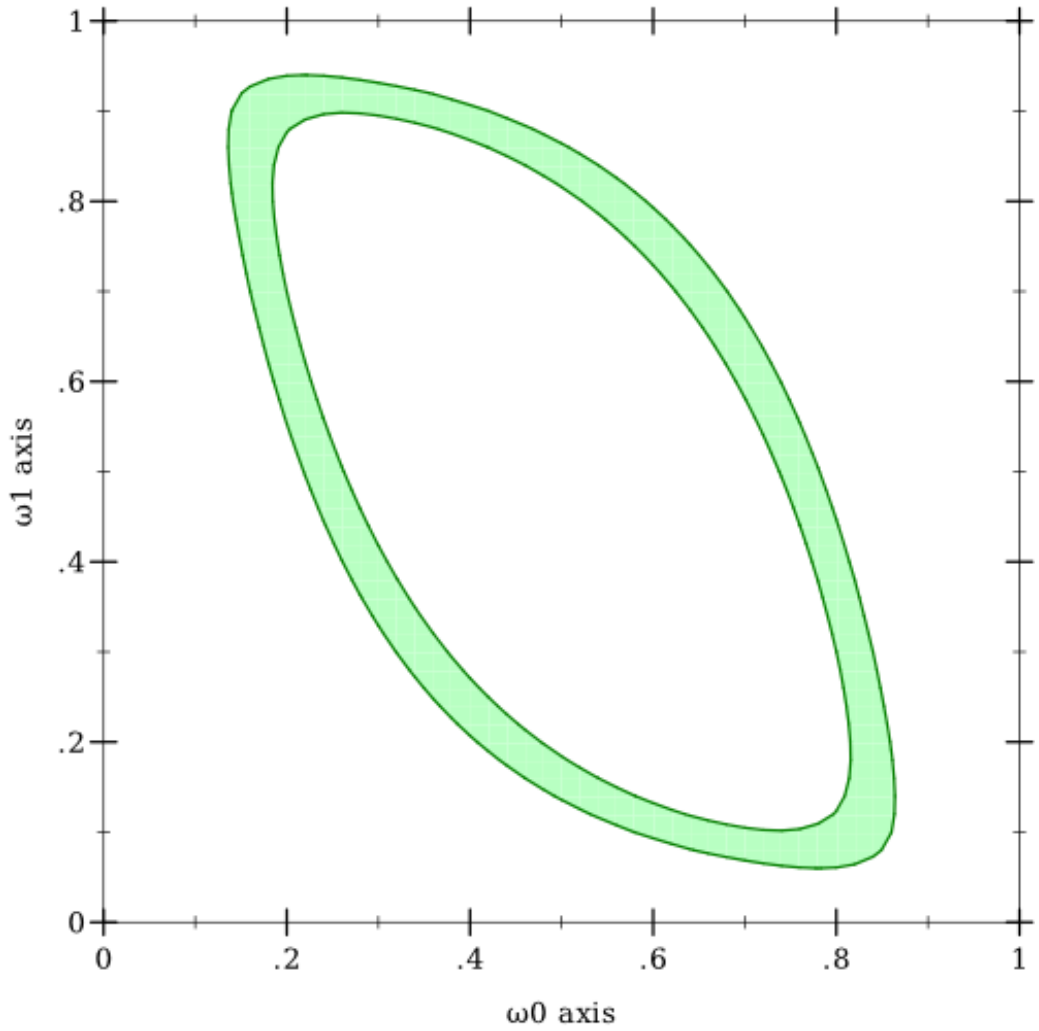
- Seems like we need:
  - Standard interpretation of programs as pure functions from a random source
  - Efficient way to compute preimage sets
  - Efficient representation of arbitrary sets
  - Efficient way to compute volumes of preimage sets
  - Proof of correctness w.r.t. standard interpretation

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 volumes of preimage sets
  - Proof of correctness w.r.t. standard interpretation
- Exact implementation is out of reach, but...

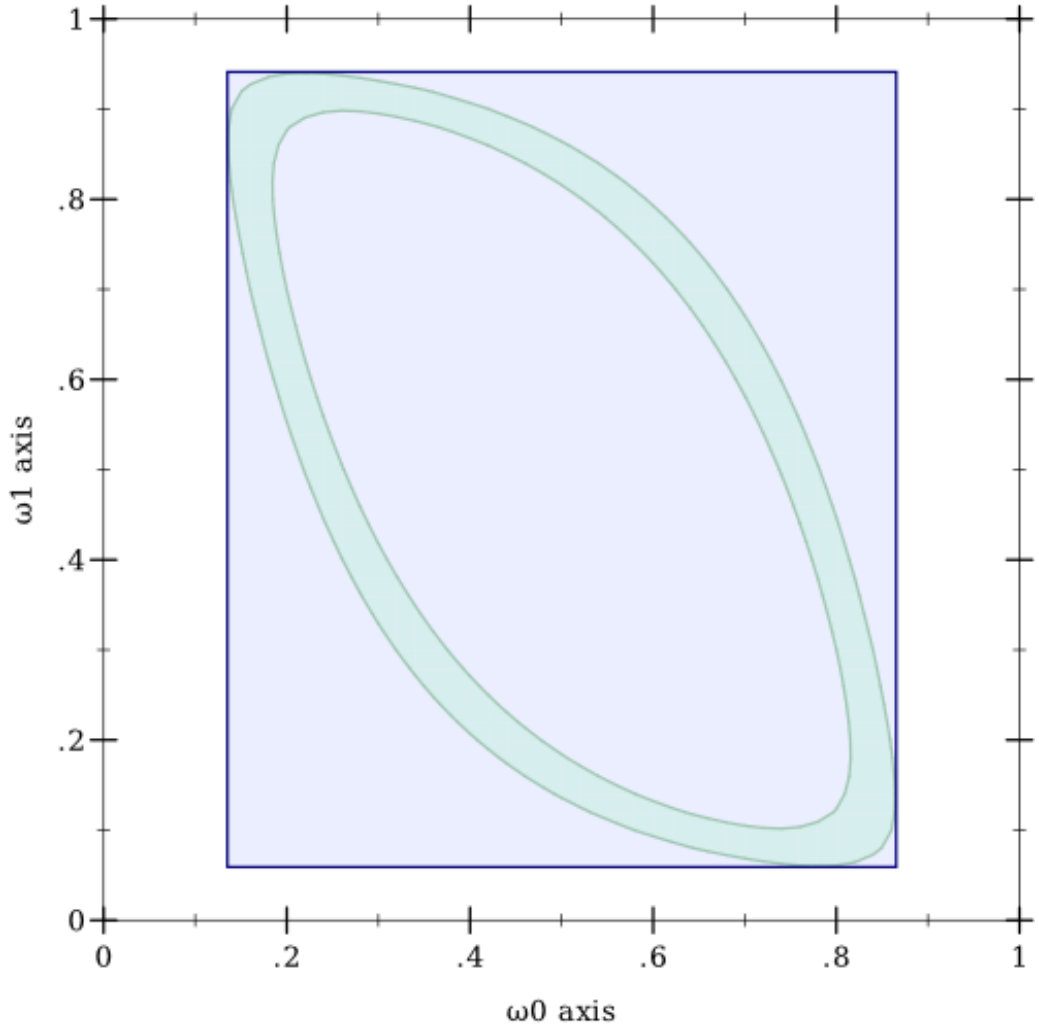
What About Approximating?

Conservative approximation with rectangles:



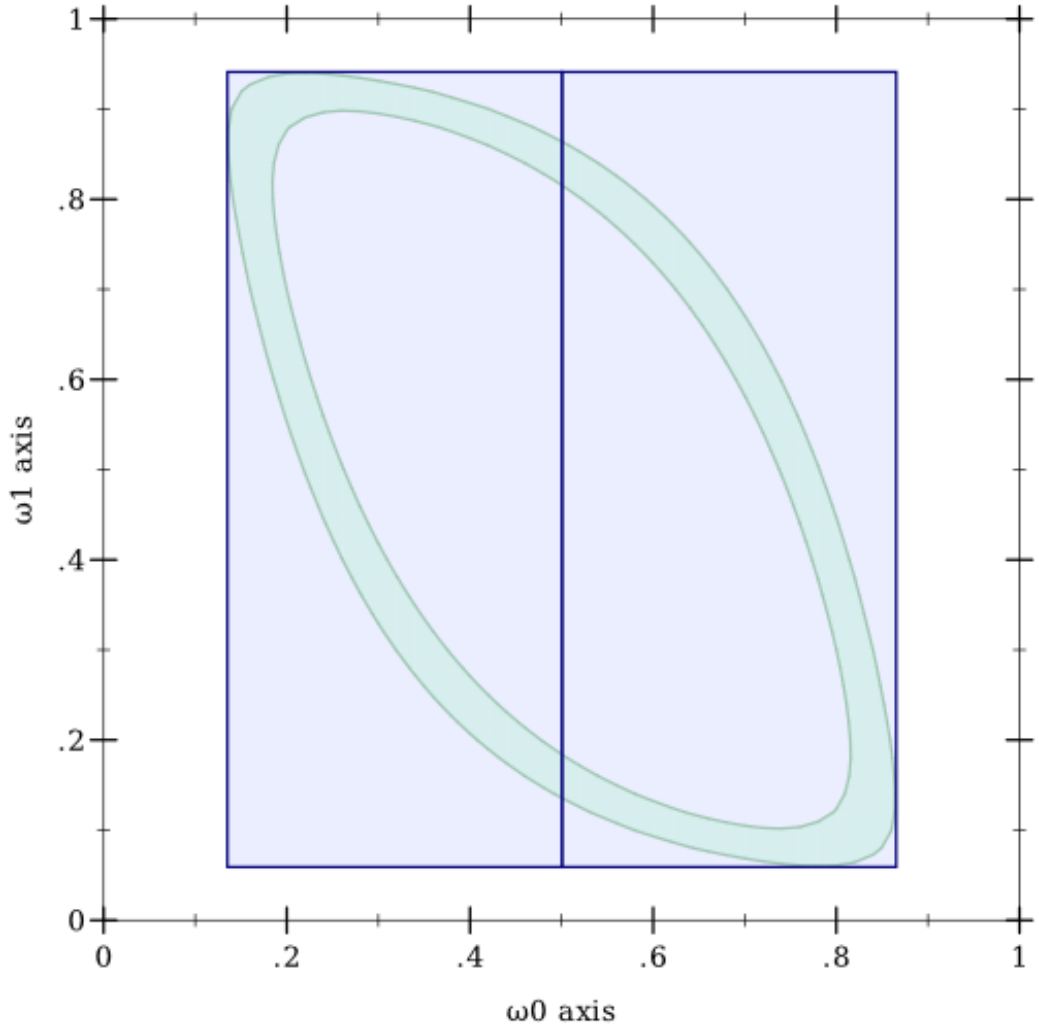
What About Approximating?

Conservative approximation with rectangles:



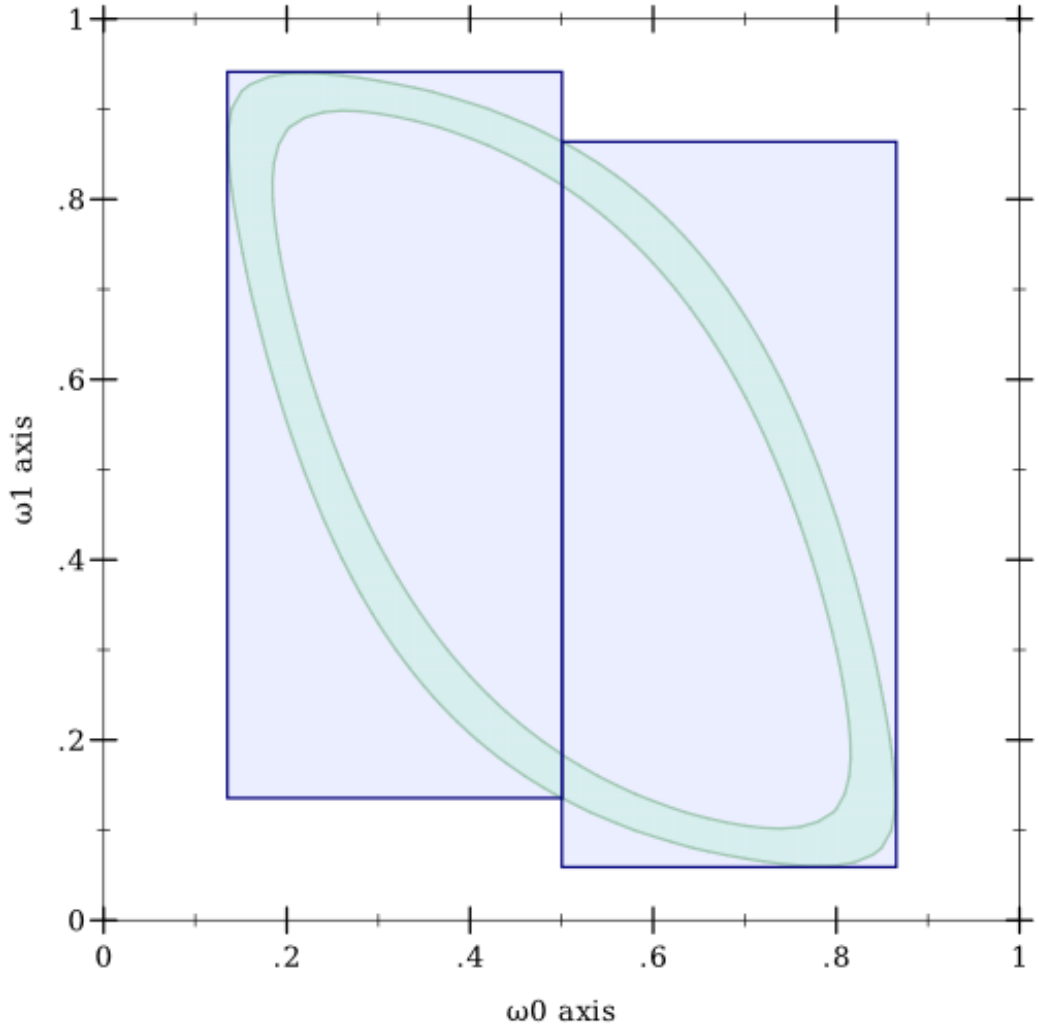
What About Approximating?

Restricting preimages to rectangular subdomains:



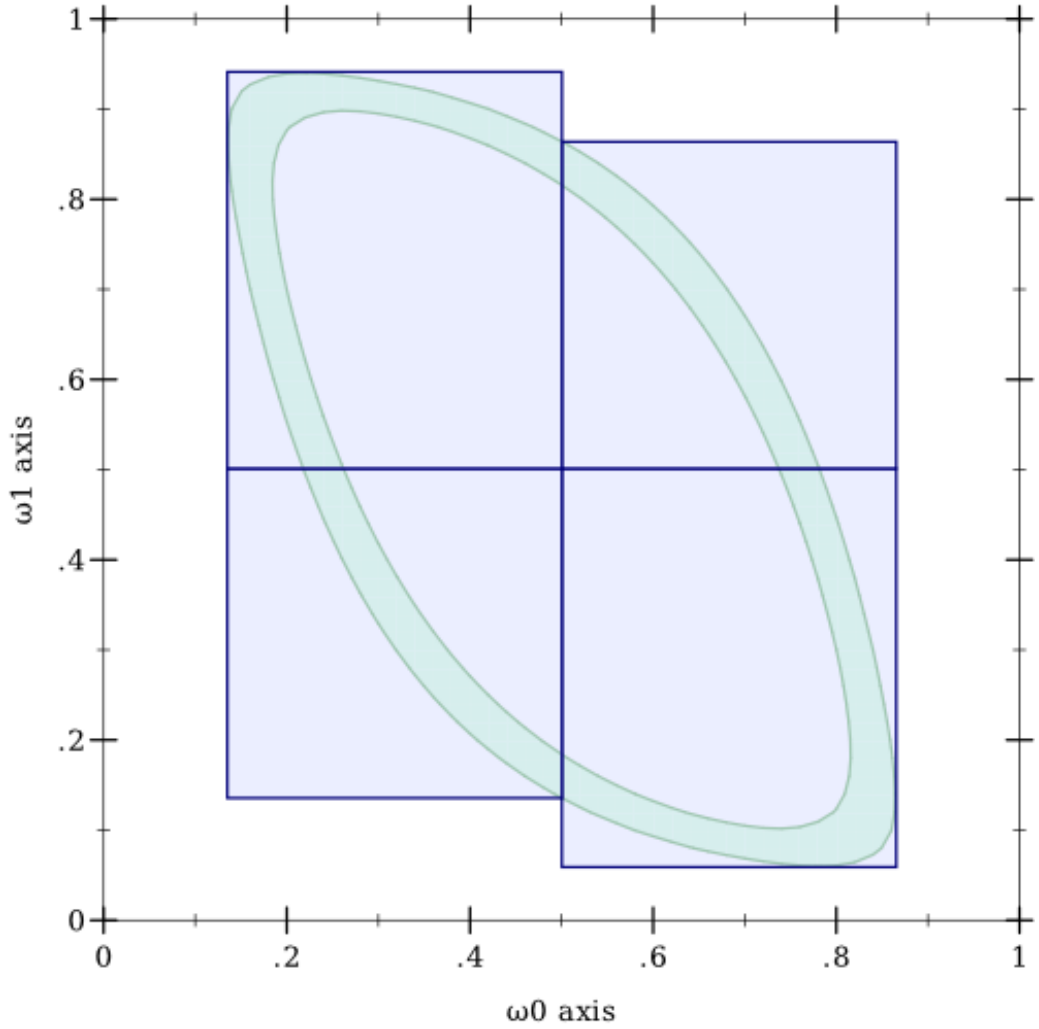
What About Approximating?

Restricting preimages to rectangular subdomains:



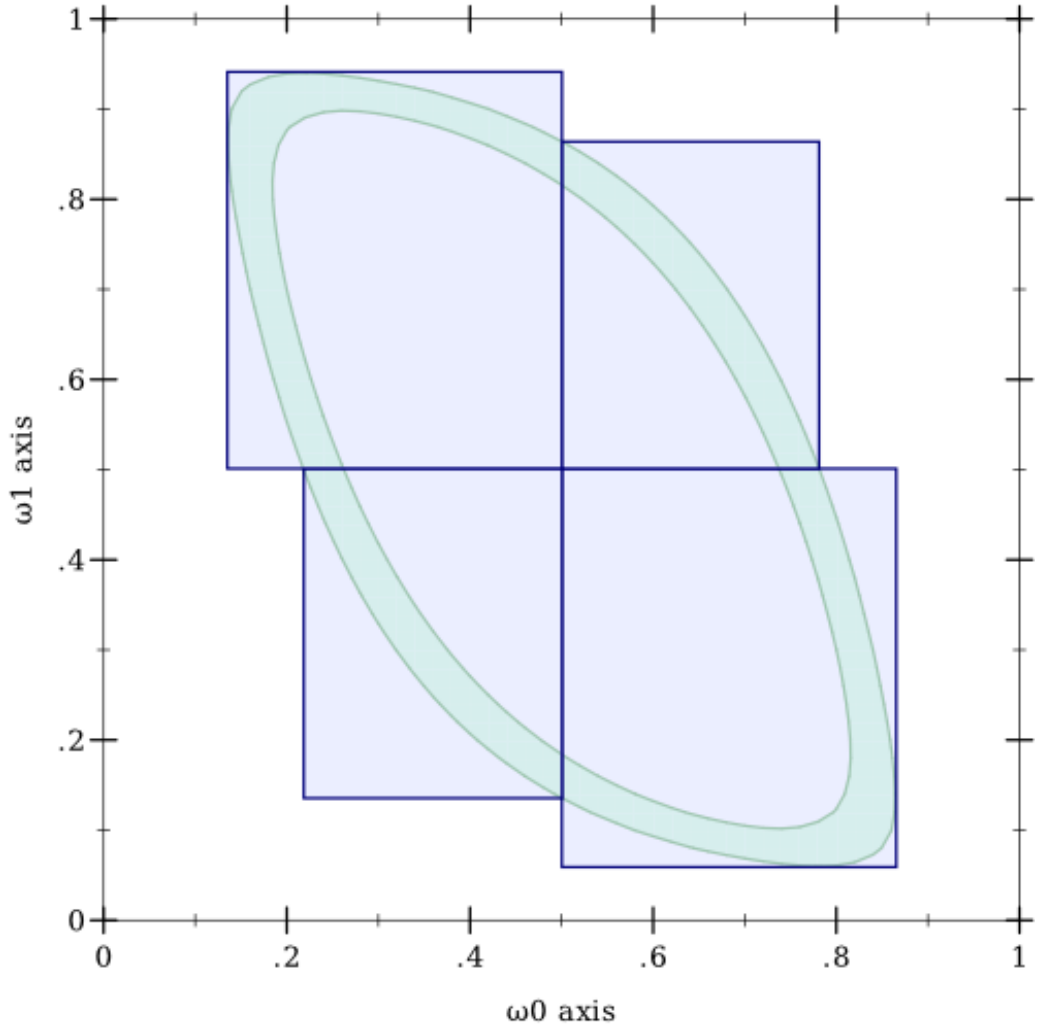
What About Approximating?

Restricting preimages to rectangular subdomains:



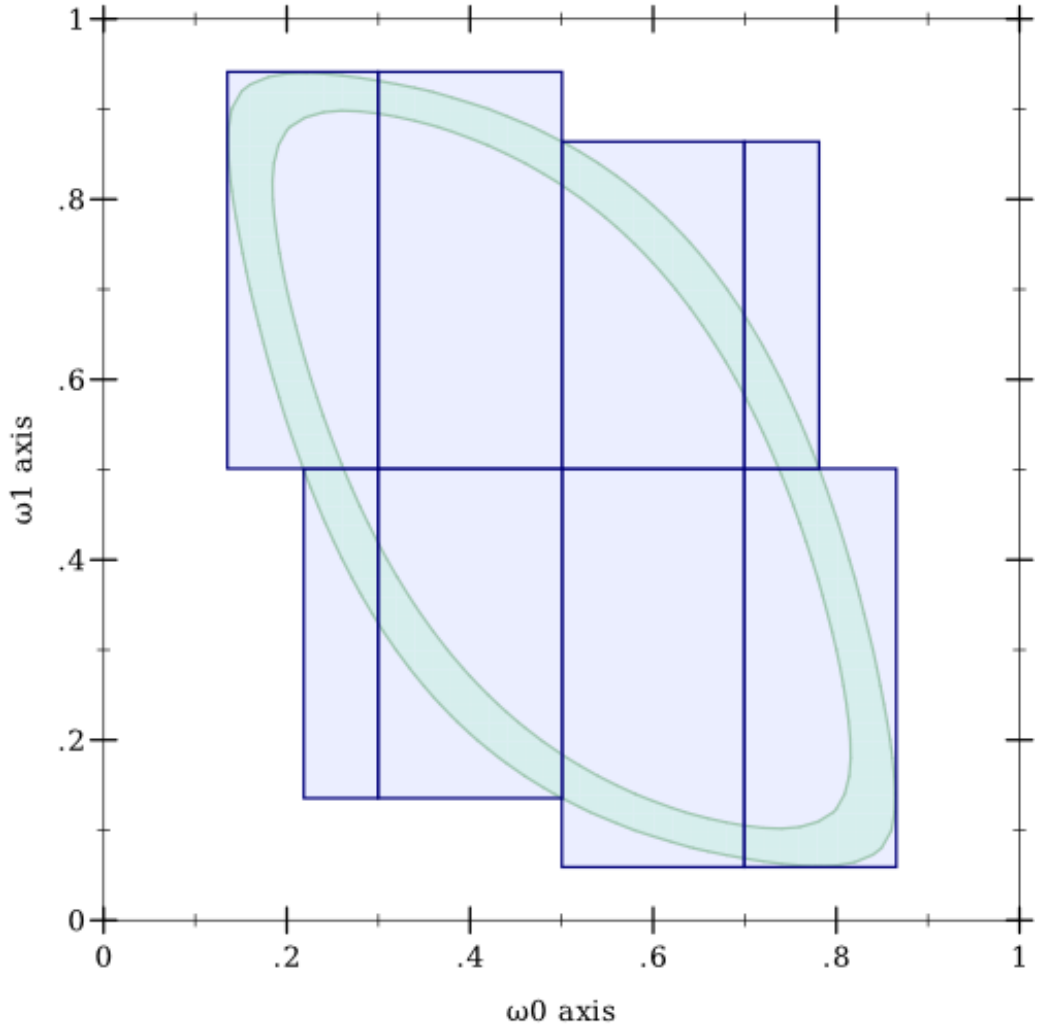
What About Approximating?

Restricting preimages to rectangular subdomains:



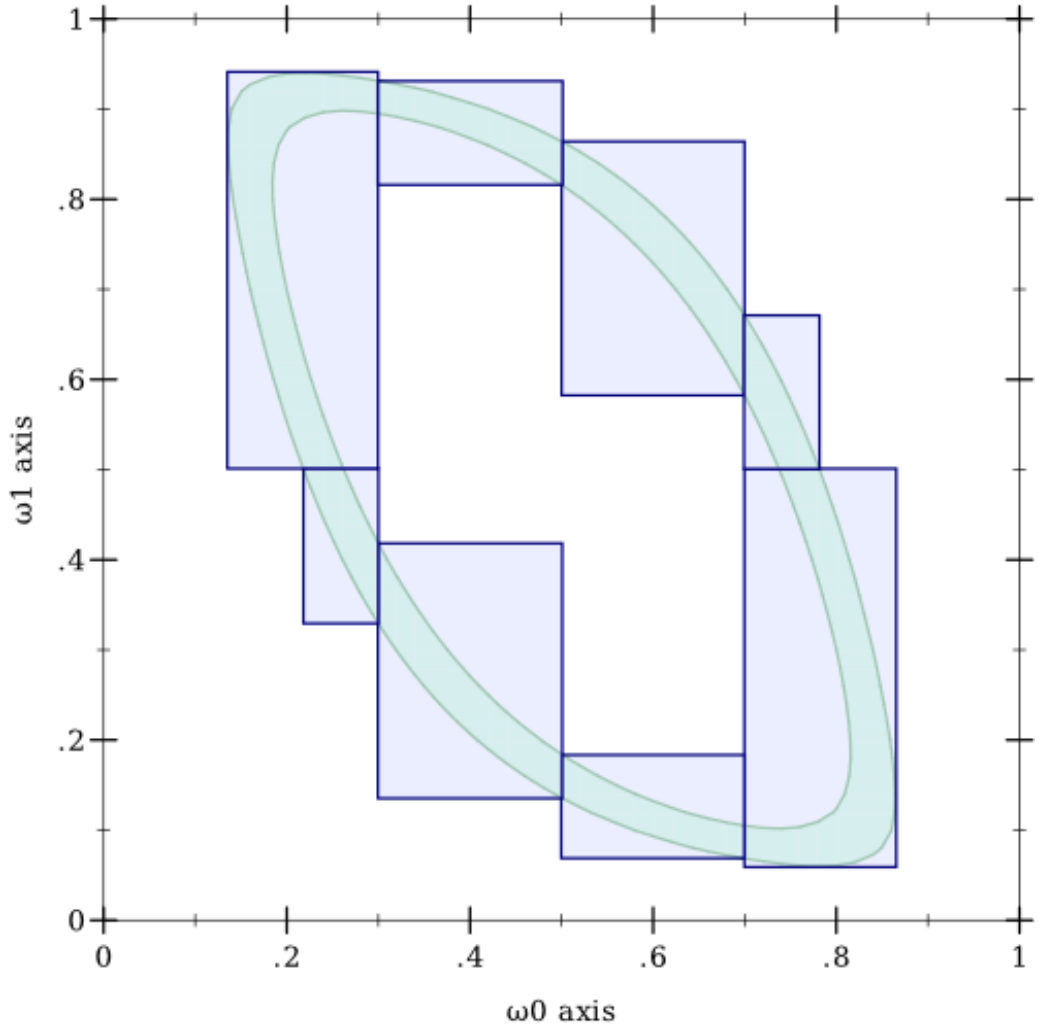
What About Approximating?

Restricting preimages to rectangular subdomains:



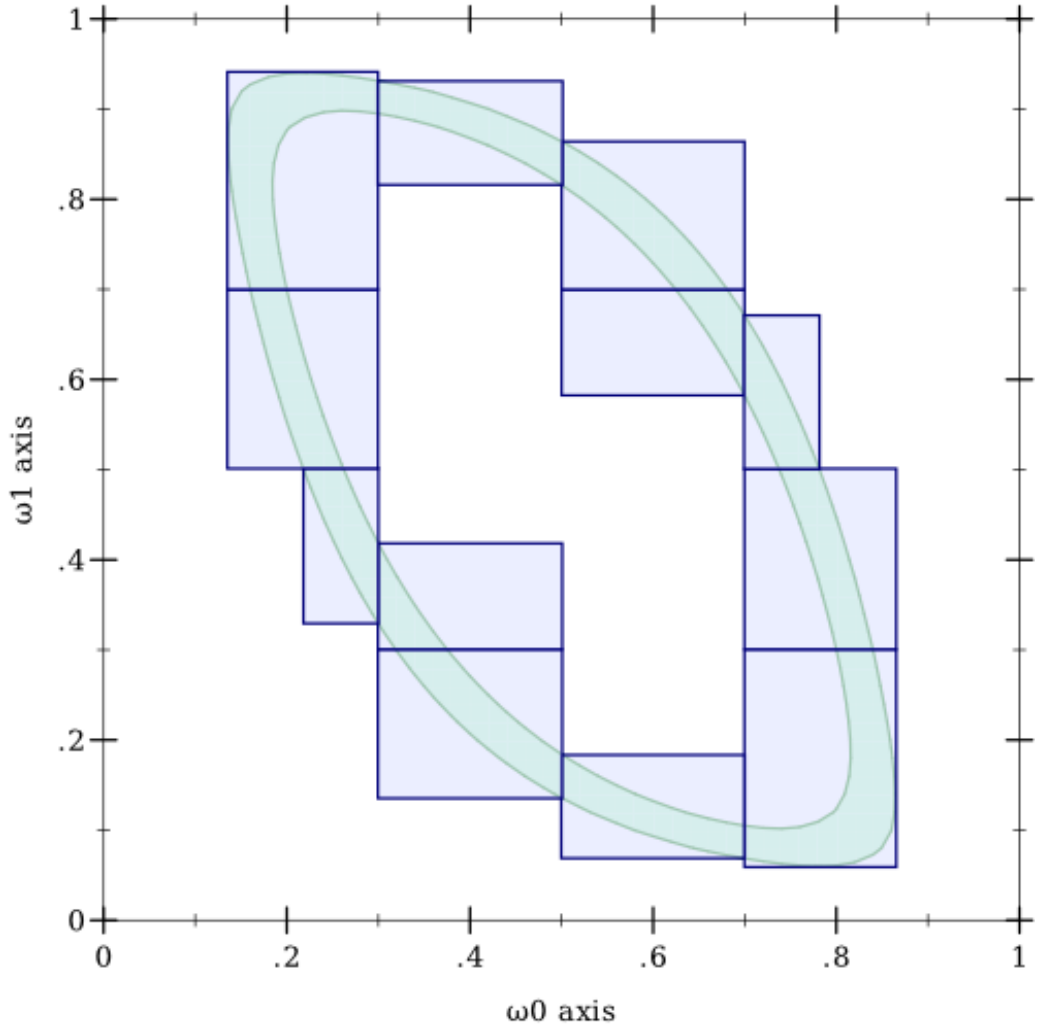
What About Approximating?

Restricting preimages to rectangular subdomains:



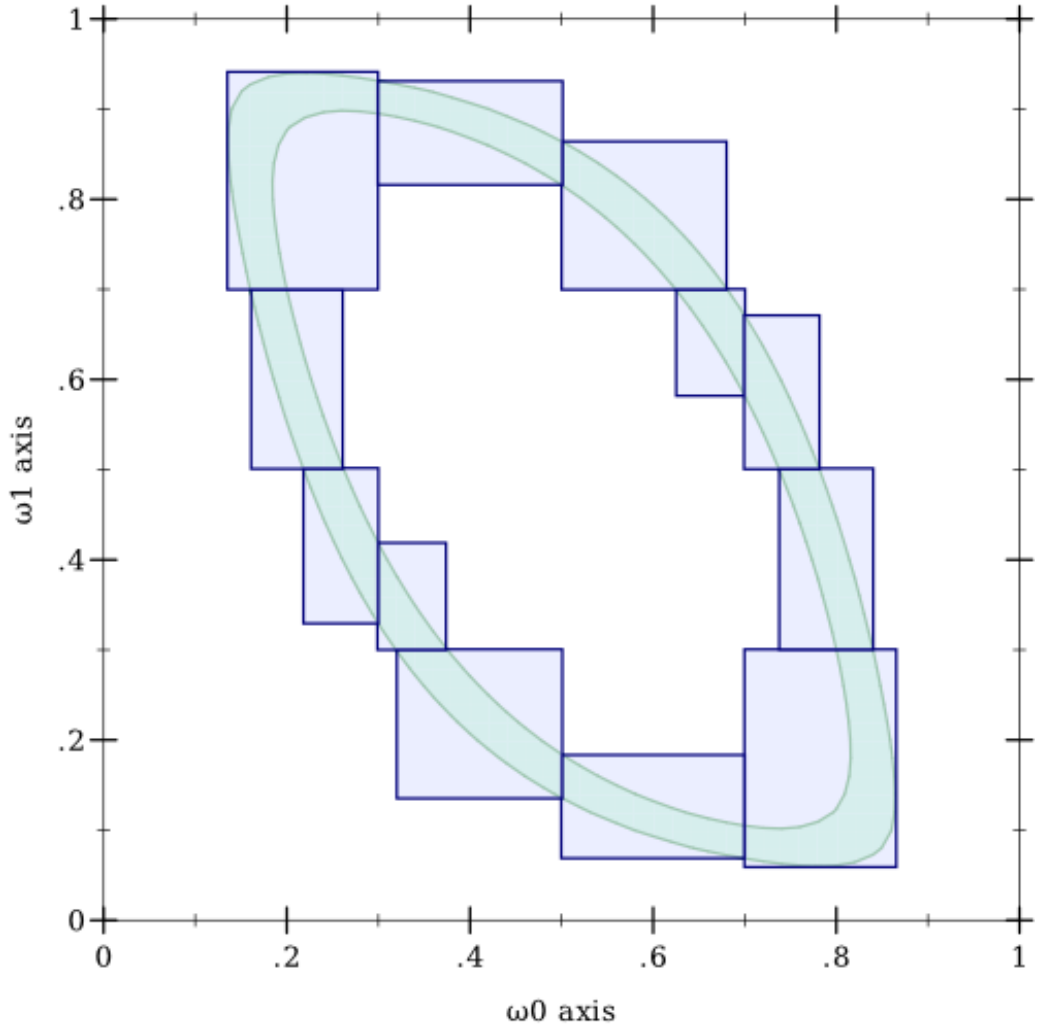
What About Approximating?

Restricting preimages to rectangular subdomains:



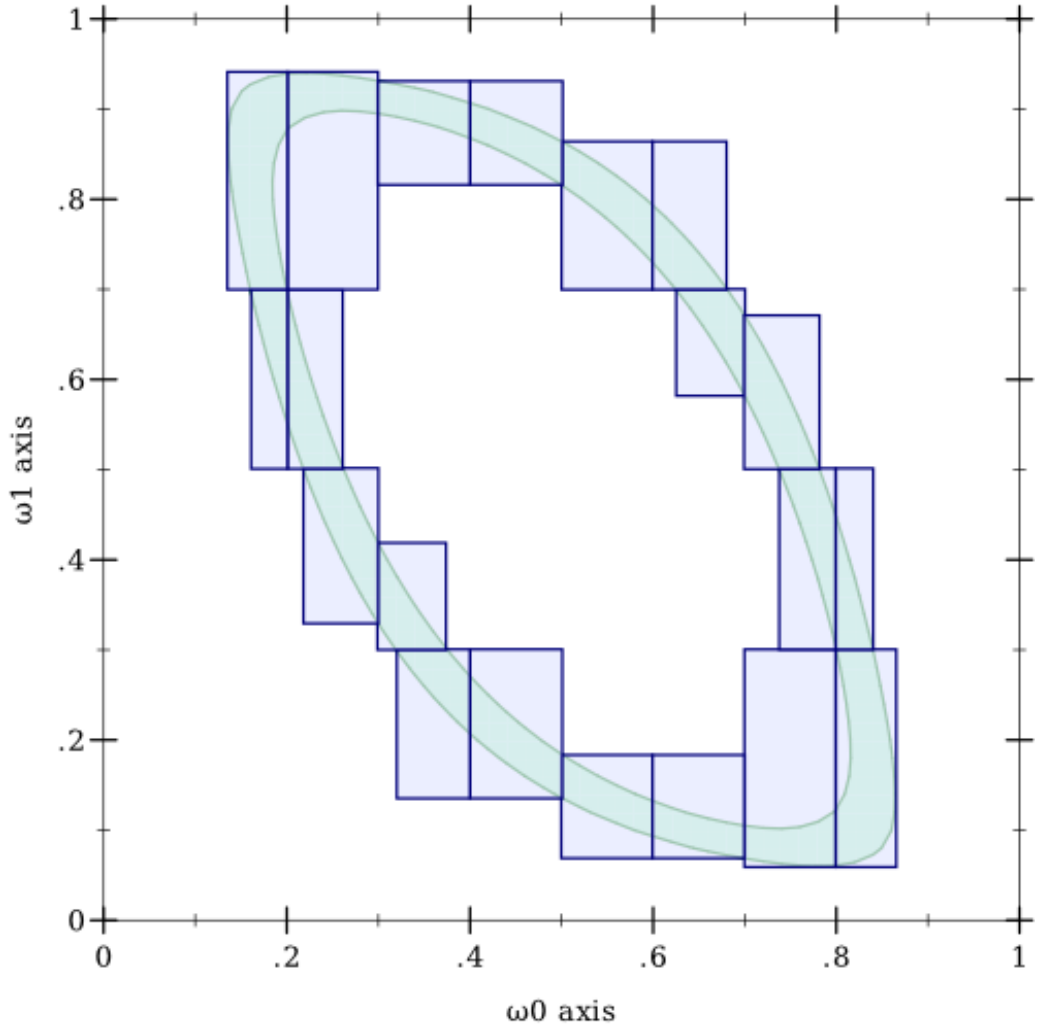
What About Approximating?

Restricting preimages to rectangular subdomains:



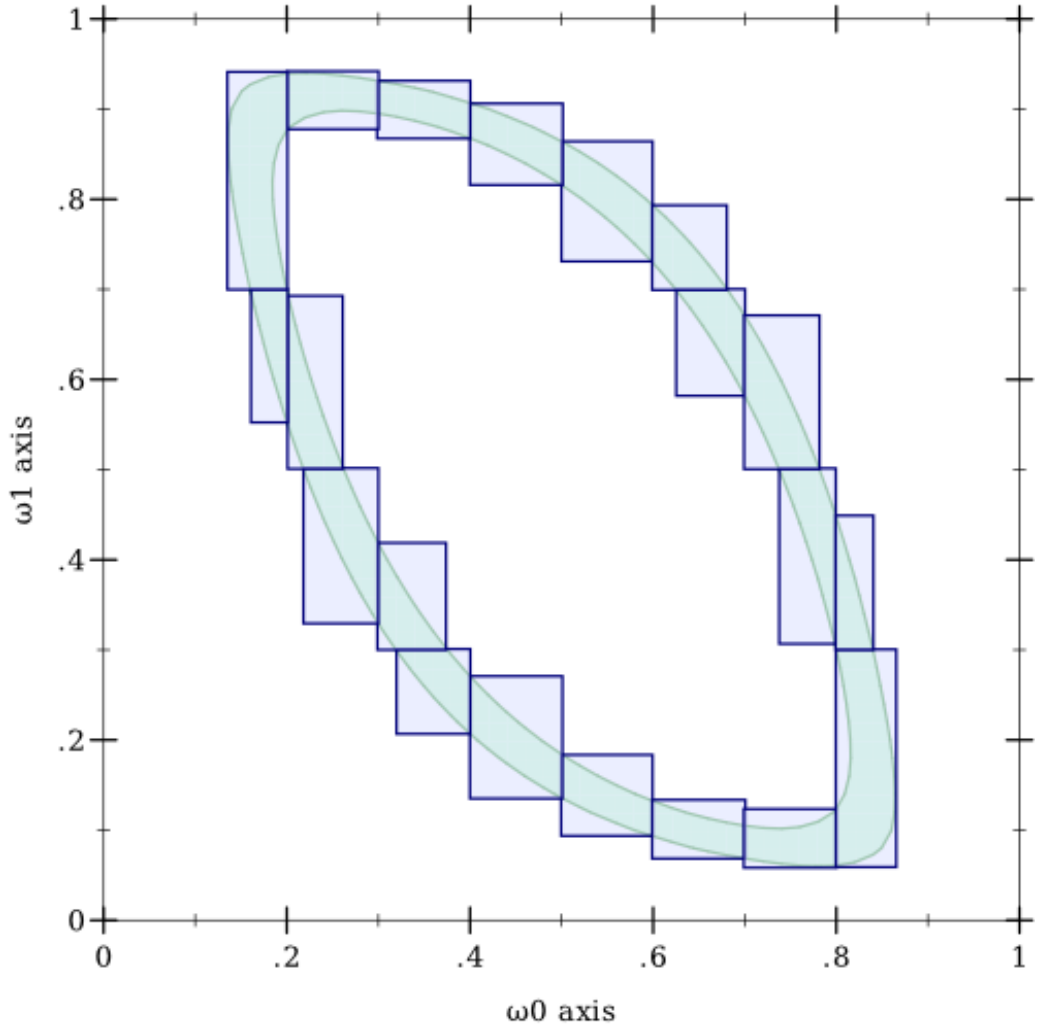
What About Approximating?

Restricting preimages to rectangular subdomains:



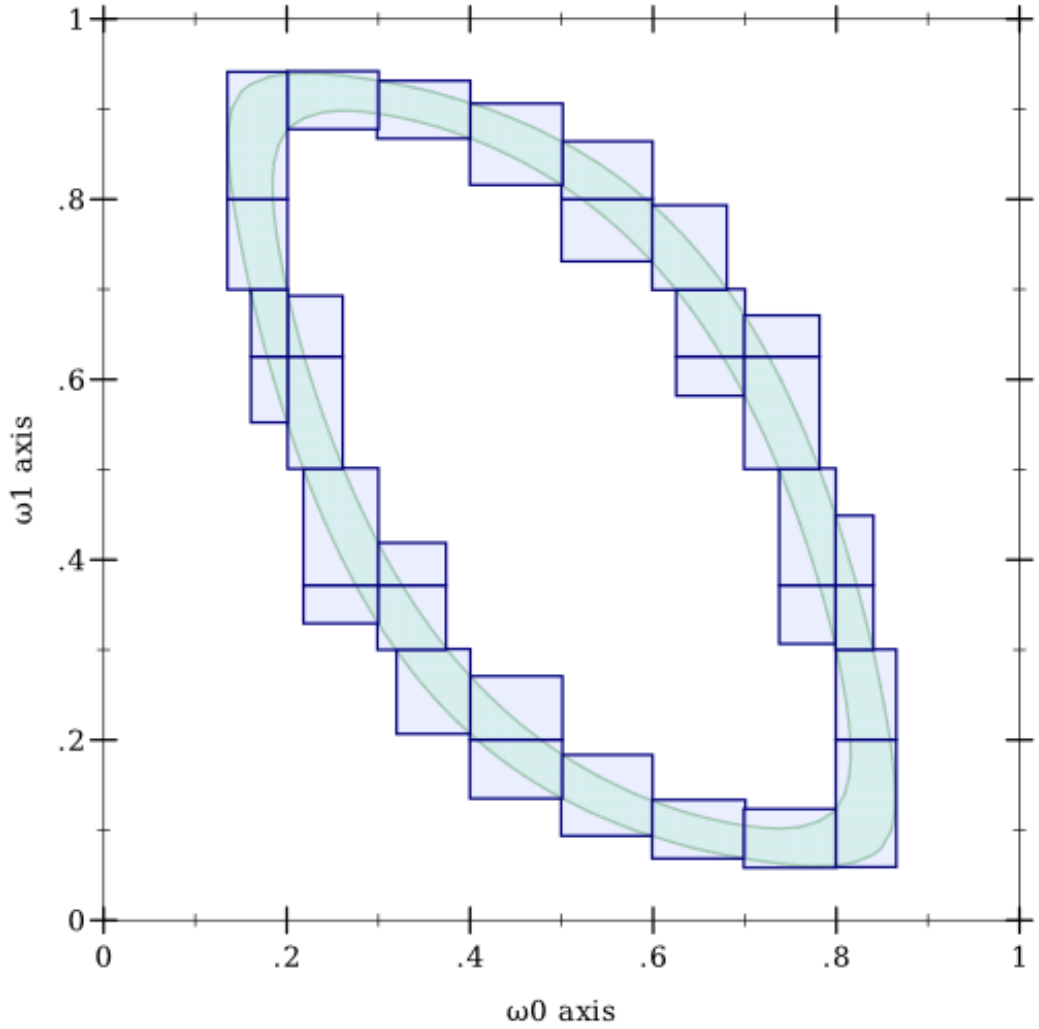
What About Approximating?

Restricting preimages to rectangular subdomains:



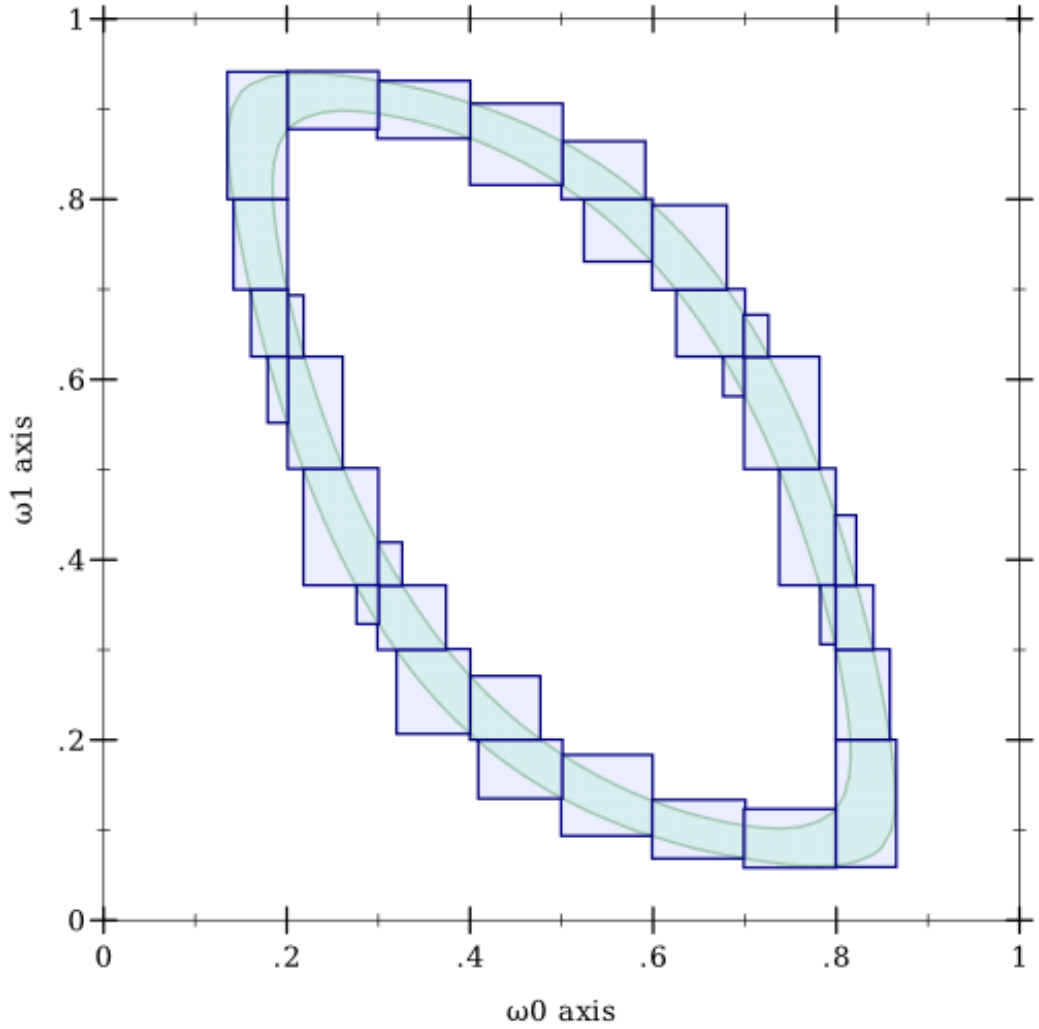
What About Approximating?

Restricting preimages to rectangular subdomains:



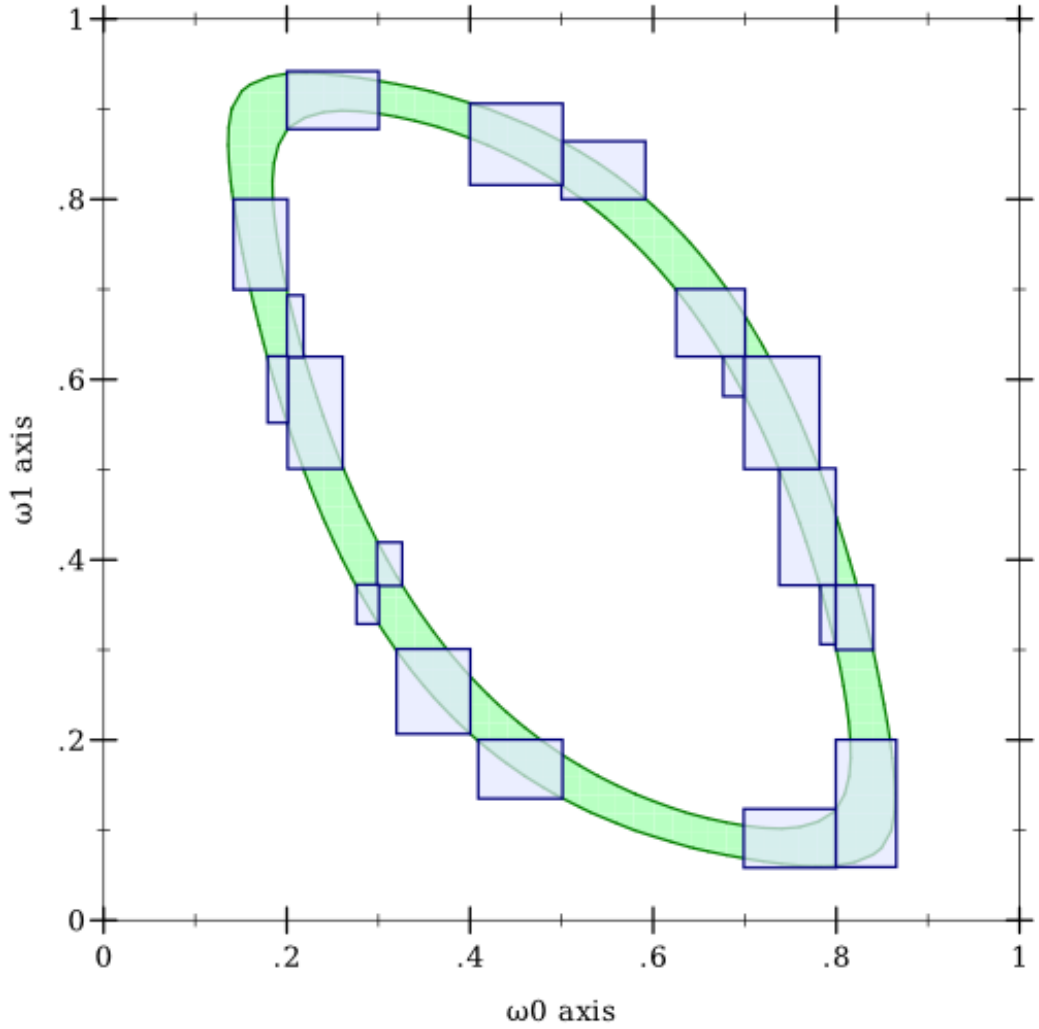
What About Approximating?

Restricting preimages to rectangular subdomains:



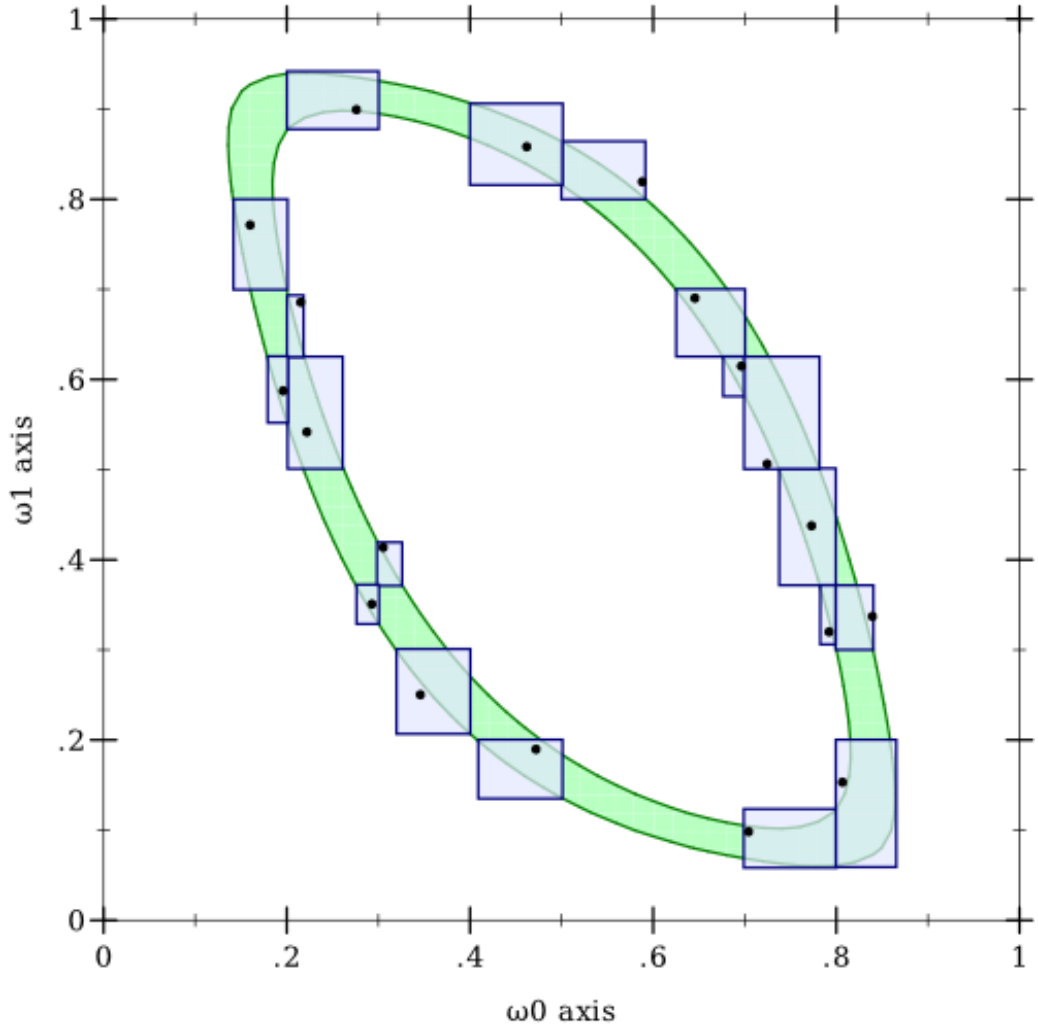
What About Approximating?

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:

$$
\begin{array}{rcl}
p & ::= & x := e;\ \cdots\ ;\ x := e;\ e \\
e & ::= & x\ e \,|\, \text{if}\ e\ e\ e \,|\, \text{let}\ e\ e \,|\, \text{env}\ n \,|\, \langle e, e \rangle \,|\, \delta\ e \,|\, v \\
x & ::= & [\text{first-order function names}] \\
\delta & ::= & [\text{primitive function names}] \\
v & ::= & [\text{first-order values}]
\end{array}
$$

Standard Interpretation

- Grammar:

$$
\begin{aligned}
p &::= x := e;\ \cdots\ ;\ x := e;\ e \\
e &::= x\ e \mid \text{if}\ e\ e\ e \mid \text{let}\ e\ e \mid \text{env}\ n \mid \langle e, e \rangle \mid \delta\ e \mid v \\
x &::= [\text{first-order function names}] \\
\delta &::= [\text{primitive function names}] \\
v &::= [\text{first-order values}]
\end{aligned}
$$

- Semantic function $[\![\cdot]\!] : p \to (\Omega \to B)$

Standard Interpretation

- Grammar:

$$
\begin{aligned}
p &::= x := e;\ \cdots\ ;\ x := e;\ e \\
e &::= x\ e \mid \text{if}\ e\ e\ e \mid \text{let}\ e\ e \mid \text{env}\ n \mid \langle e, e \rangle \mid \delta\ e \mid v \\
x &::= [\text{first-order function names}] \\
\delta &::= [\text{primitive function names}] \\
v &::= [\text{first-order values}]
\end{aligned}
$$

- Semantic function $[\![\cdot]\!] : p \to (\Omega \to B)$
- Math has no general recursion, so $[\![p]\!]$ (i.e. interpretation of program p) is a *λ-calculus* term

Standard Interpretation

- Grammar:

$$
\begin{aligned}
p &::= x := e;\ \cdots\ ;\ x := e;\ e \\
e &::= x\ e \mid \text{if}\ e\ e\ e \mid \text{let}\ e\ e \mid \text{env}\ n \mid \langle e, e \rangle \mid \delta\ e \mid v \\
x &::= [\text{first-order function names}] \\
\delta &::= [\text{primitive function names}] \\
v &::= [\text{first-order values}]
\end{aligned}
$$

- Semantic function $[\![\cdot]\!] : p \to (\Omega \to B)$
- Math has no general recursion, so $[\![p]\!]$ (i.e. interpretation of program p) is a *λ-calculus* term
- Easy implementation in any language with lambdas

Compositional Semantics

- Compositional: every expression has a meaning independent of surrounding expressions

Compositional Semantics

- Compositional: every expression has a meaning independent of surrounding expressions
- Example:

$$[\![\langle e_1, e_2 \rangle]\!] \;=\; \mathrm{pair}\; [\![e_1]\!]\; [\![e_2]\!]$$

Compositional Semantics

- Compositional: every expression has a meaning independent of surrounding expressions
- Example:

$$[\![\langle e_1, e_2 \rangle]\!] \;=\; \text{pair}\; [\![e_1]\!]\; [\![e_2]\!]$$

where

$$\text{pair} : (A \to B_1) \to (A \to B_2) \to (A \to \langle B_1, B_2 \rangle)$$
$$\text{pair}\; f_1\; f_2 \;=\; \lambda\, a.\, \langle f_1\; a, f_2\; a \rangle$$

Compositional Semantics

- Compositional: every expression has a meaning independent of surrounding expressions
- Example:

$$[\![\langle e_1, e_2 \rangle]\!] \;=\; \mathrm{pair}\; [\![e_1]\!]\; [\![e_2]\!]$$

where

$$\mathrm{pair} : (A \to B_1) \to (A \to B_2) \to (A \to \langle B_1, B_2 \rangle)$$
$$\mathrm{pair}\;\; f_1\;\; f_2 \;=\; \lambda\, a.\, \langle f_1\;\, a, f_2\;\, a \rangle$$

- Advantage: proofs about all programs by structural induction

Compositional Semantics

- Compositional: every expression has a meaning independent of surrounding expressions
- Example:

$$[\![\langle e_1, e_2 \rangle]\!] \;=\; \mathrm{pair}\; [\![e_1]\!]\; [\![e_2]\!]$$

where

$$\mathrm{pair} : (A \to B_1) \to (A \to B_2) \to (A \to \langle B_1, B_2 \rangle)$$
$$\mathrm{pair}\;\; f_1\;\; f_2 \;=\; \lambda\, a.\, \langle f_1\;\, a, f_2\;\, a \rangle$$

- Advantage: proofs about all programs by structural induction
- Can preimages be computed compositionally?

Pair Preimages

$$f_1(\omega) = \omega_0 + \omega_1 \qquad f_2(\omega) = \omega_0 \cdot \omega_1$$

Pair Preimages

$$f_1(\omega) = \omega_0 + \omega_1 \qquad f_2(\omega) = \omega_0 \cdot \omega_1$$

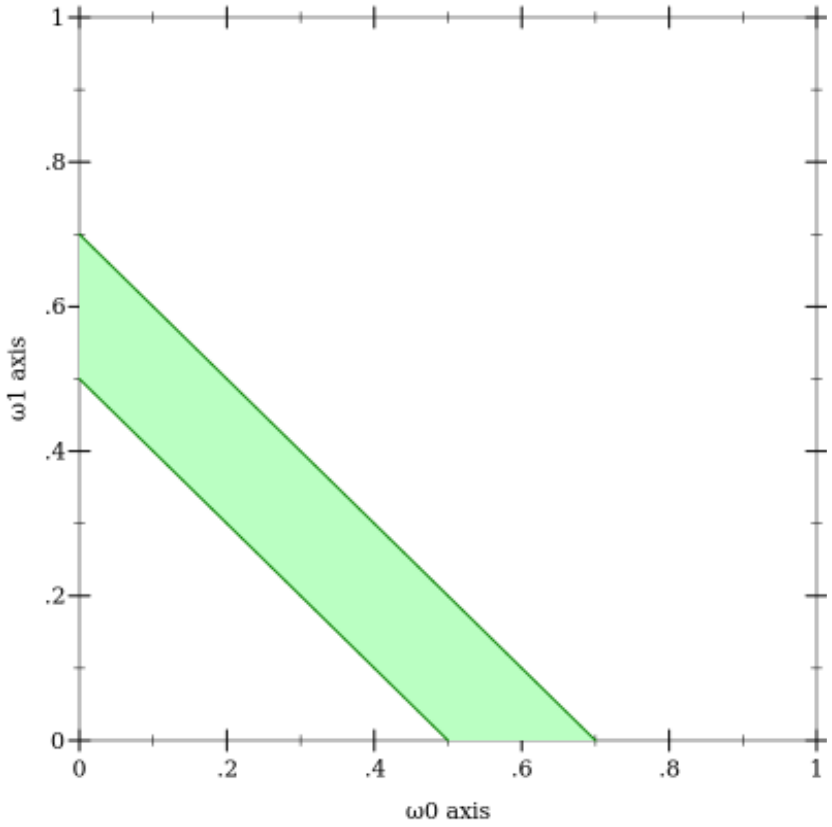
$$f = \text{pair } f_1 \; f_2 = \lambda\, \omega.\, \langle \omega_0 + \omega_1, \omega_0 \cdot \omega_1 \rangle$$

Pair Preimages

$$f_1(\omega) = \omega_0 + \omega_1 \qquad f_2(\omega) = \omega_0 \cdot \omega_1$$

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

$f_1^{-1}([0.5, 0.7])$:

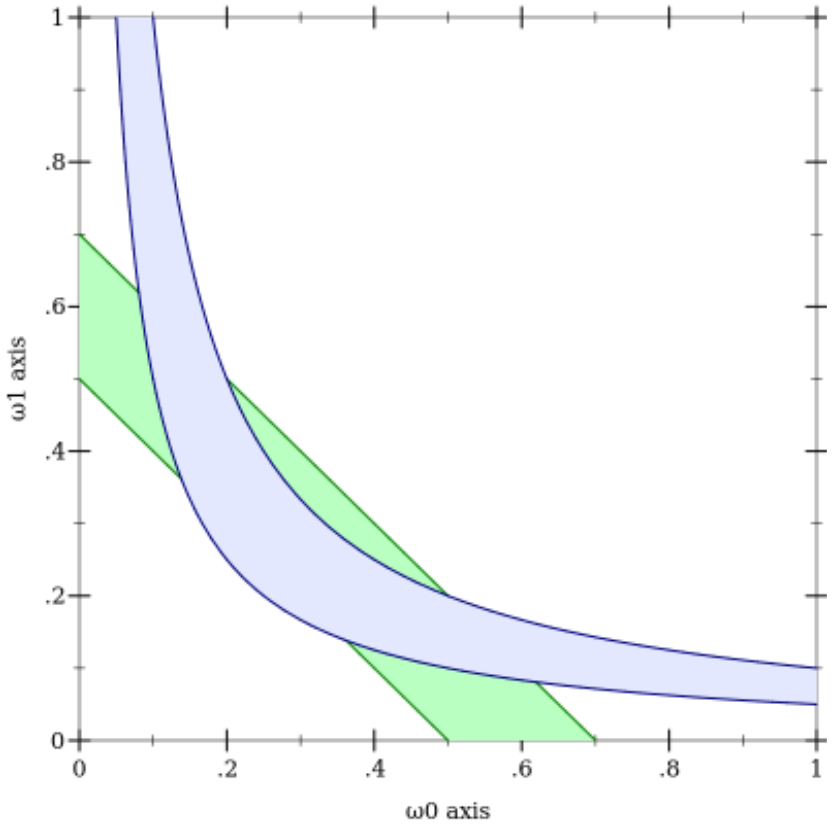


Pair Preimages

$$f_1(\omega) = \omega_0 + \omega_1 \qquad f_2(\omega) = \omega_0 \cdot \omega_1$$

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

$f_1^{-1}([0.5, 0.7])$ and $f_2^{-1}([0.05, 0.1])$:

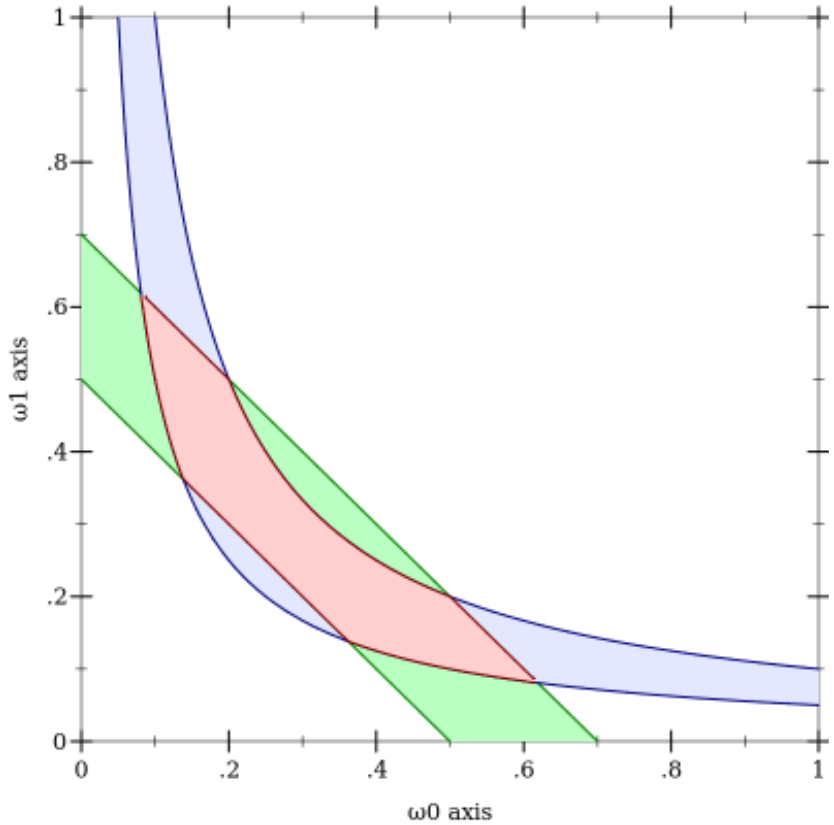


Pair Preimages

$$f_1(\omega) = \omega_0 + \omega_1 \qquad f_2(\omega) = \omega_0 \cdot \omega_1$$

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

$f_1^{-1}([0.5, 0.7])$ and $f_2^{-1}([0.05, 0.1])$ and $f^{-1}([0.5, 0.7] \times [0.05, 0.1])$:



Nonstandard Interpretation: Computing Preimages

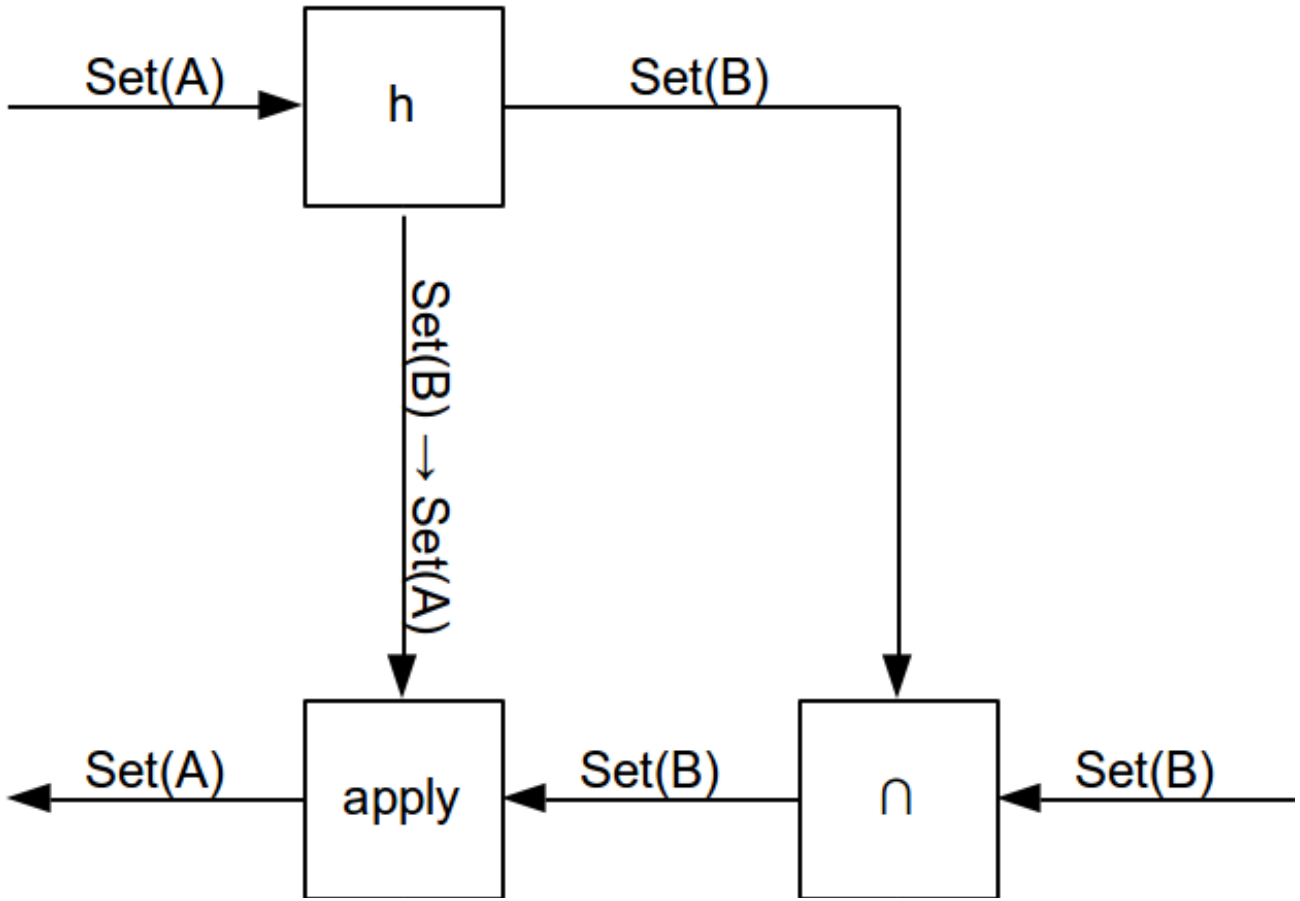
- Preimage computation:

$$A \underset{\text{pre}}{\rightsquigarrow} B ::= \text{Set}(A) \to \langle \text{Set}(B), \text{Set}(B) \to \text{Set}(A) \rangle$$

Nonstandard Interpretation: Computing Preimages

- Preimage computation:

$$A \underset{\text{pre}}{\rightsquigarrow} B ::= \text{Set}(A) \to \langle \text{Set}(B), \text{Set}(B) \to \text{Set}(A) \rangle$$



Nonstandard Interpretation: Preimages Under Pairing

- Pairing types:

$$\mathrm{pair} : (A \to B_1) \to (A \to B_2) \to (A \to \langle B_1, B_2 \rangle)$$

$$\mathrm{pair}_{\mathrm{pre}} : (A \overset{\sim}{\underset{\mathrm{pre}}{\to}} B_1) \to (A \overset{\sim}{\underset{\mathrm{pre}}{\to}} B_2) \to (A \overset{\sim}{\underset{\mathrm{pre}}{\to}} \langle B_1, B_2 \rangle)$$

Nonstandard Interpretation: Preimages Under Pairing

- Pairing types:

$$\mathrm{pair} : (A \to B_1) \to (A \to B_2) \to (A \to \langle B_1, B_2 \rangle)$$
$$\mathrm{pair}_{\mathrm{pre}} : (A \overset{\sim}{\underset{\mathrm{pre}}{\to}} B_1) \to (A \overset{\sim}{\underset{\mathrm{pre}}{\to}} B_2) \to (A \overset{\sim}{\underset{\mathrm{pre}}{\to}} \langle B_1, B_2 \rangle)$$

Theorem (correctness under pairing). If

- $h_1 : A \overset{\sim}{\underset{\mathrm{pre}}{\to}} B_1$ computes preimages under $f_1 : A \to B_1$

Nonstandard Interpretation: Preimages Under Pairing

- Pairing types:

$$\mathrm{pair} : (A \to B_1) \to (A \to B_2) \to (A \to \langle B_1, B_2 \rangle)$$
$$\mathrm{pair}_{\mathrm{pre}} : (A \overset{\sim}{\underset{\mathrm{pre}}{\to}} B_1) \to (A \overset{\sim}{\underset{\mathrm{pre}}{\to}} B_2) \to (A \overset{\sim}{\underset{\mathrm{pre}}{\to}} \langle B_1, B_2 \rangle)$$

Theorem (correctness under pairing). If

- $h_1 : A \overset{\sim}{\underset{\mathrm{pre}}{\to}} B_1$ computes preimages under $f_1 : A \to B_1$
- $h_2 : A \overset{\sim}{\underset{\mathrm{pre}}{\to}} B_2$ computes preimages under $f_2 : A \to B_2$

Nonstandard Interpretation: Preimages Under Pairing

- Pairing types:

$$\mathrm{pair} : (A \to B_1) \to (A \to B_2) \to (A \to \langle B_1, B_2 \rangle)$$
$$\mathrm{pair}_{\mathrm{pre}} : (A \underset{\mathrm{pre}}{\rightsquigarrow} B_1) \to (A \underset{\mathrm{pre}}{\rightsquigarrow} B_2) \to (A \underset{\mathrm{pre}}{\rightsquigarrow} \langle B_1, B_2 \rangle)$$

Theorem (correctness under pairing). If

- $h_1 : A \underset{\mathrm{pre}}{\rightsquigarrow} B_1$ computes preimages under $f_1 : A \to B_1$
- $h_2 : A \underset{\mathrm{pre}}{\rightsquigarrow} B_2$ computes preimages under $f_2 : A \to B_2$

then $\mathrm{pair}_{\mathrm{pre}}\ h_1\ h_2$ computes preimages under $\mathrm{pair}\ f_1\ f_2$.

Nonstandard Interpretation: Preimages Under Pairing

- Pairing types:

$$\mathrm{pair} : (A \to B_1) \to (A \to B_2) \to (A \to \langle B_1, B_2 \rangle)$$
$$\mathrm{pair}_{\mathrm{pre}} : (A \underset{\mathrm{pre}}{\rightsquigarrow} B_1) \to (A \underset{\mathrm{pre}}{\rightsquigarrow} B_2) \to (A \underset{\mathrm{pre}}{\rightsquigarrow} \langle B_1, B_2 \rangle)$$

Theorem (correctness under pairing). If

- $h_1 : A \underset{\mathrm{pre}}{\rightsquigarrow} B_1$ computes preimages under $f_1 : A \to B_1$
- $h_2 : A \underset{\mathrm{pre}}{\rightsquigarrow} B_2$ computes preimages under $f_2 : A \to B_2$

then $\mathrm{pair}_{\mathrm{pre}}\ h_1\ h_2$ computes preimages under $\mathrm{pair}\ f_1\ f_2$.

- Similar theorems for every kind of expression

Nonstandard Interpretation: Correctness

Theorem. For all programs p, $[\![p]\!]_{\mathrm{pre}}$ computes preimages under $[\![p]\!]$.

Proof. By structural induction on program terms.

Wait a Minute

- Q. Don’t the interpretations of $[\![\cdot]\!]_{\mathrm{pre}}$ do uncomputable things?

Wait a Minute

- Q. Don’t the interpretations of $[\![\cdot]\!]_{\mathrm{pre}}$ do uncomputable things?
  - A. Yes. Yes, they do.

Wait a Minute

- Q. Don't the interpretations of $[\![\cdot]\!]_{\mathrm{pre}}$ do uncomputable things?
  - A. Yes. Yes, they do.
- Q. Where do I get a computer that runs them?

Wait a Minute

- Q. Don't the interpretations of $[\![\cdot]\!]_{\mathrm{pre}}$ do uncomputable 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 $[\![\cdot]\!]_{\mathrm{pre}}$ do uncomputable 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. Where did you get a λ-calculus that could operate on arbitrary, possibly infinite sets, anyway?
  - A. Well...

Lambda-ZFC

N. Toronto and J. McCarthy. Computing in Cantor's Paradise With λ_{ZFC}. FLOPS 2012

Lambda-ZFC

N. Toronto and J. McCarthy. Computing in Cantor's Paradise With λ_{ZFC}. FLOPS 2012

λ calculus

Lambda-ZFC

N. Toronto and J. McCarthy. Computing in Cantor's Paradise With λ_{ZFC}. FLOPS 2012

λ calculus

+

Infinite sets and operations

Lambda-ZFC

N. Toronto and J. McCarthy. Computing in Cantor's Paradise With λ_{ZFC}. FLOPS 2012

λ calculus

+

Infinite sets and operations

=

λ_{ZFC}

Lambda-ZFC

N. Toronto and J. McCarthy. Computing in Cantor's Paradise With λ_{ZFC}. FLOPS 2012

λ calculus

Infinite sets and operations

λ_{ZFC}

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

Lambda-ZFC

N. Toronto and J. McCarthy. Computing in Cantor's Paradise With λ_{ZFC}. FLOPS 2012

λ calculus

Infinite sets and operations

λ_{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

N. Toronto and J. McCarthy. Computing in Cantor's Paradise With λ_{ZFC}. FLOPS 2012

λ calculus + Infinite sets and operations = λ_{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 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 intersection and join (union-like) operation, empty test, other operations
- Recall:

$$A \underset{\text{pre}}{\rightsquigarrow} B ::= \text{Set}(A) \to \langle \text{Set}(B), \text{Set}(B) \to \text{Set}(A) \rangle$$

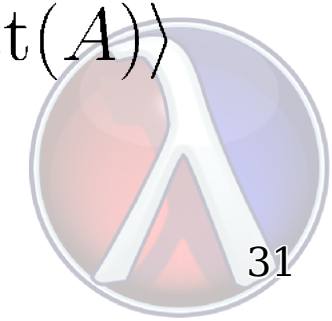
Rectangular Approximation

- A *rectangle* is
  - An interval or union of intervals
  - $A \times B$ for rectangles A and B
- Easy intersection and join (union-like) operation, empty test, other operations
- Recall:

$$A \underset{\text{pre}}{\rightsquigarrow} B ::= \text{Set}(A) \to \langle \text{Set}(B), \text{Set}(B) \to \text{Set}(A) \rangle$$

- Define:

$$A \underset{\text{pre}}{\rightsquigarrow}' B ::= \text{Rect}(A) \to \langle \text{Rect}(B), \text{Rect}(B) \to \text{Rect}(A) \rangle$$



Rectangular Approximation

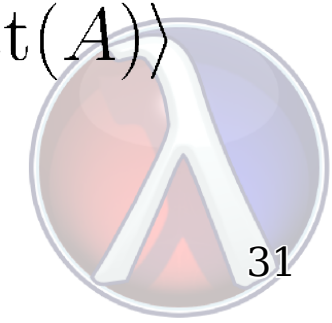
- A *rectangle* is
  - An interval or union of intervals
  - $A \times B$ for rectangles A and B
- Easy intersection and join (union-like) operation, empty test, other operations
- Recall:

$$A \underset{\text{pre}}{\rightsquigarrow} B ::= \text{Set}(A) \to \langle \text{Set}(B), \text{Set}(B) \to \text{Set}(A) \rangle$$

- Define:

$$A \underset{\text{pre}}{\rightsquigarrow}' B ::= \text{Rect}(A) \to \langle \text{Rect}(B), \text{Rect}(B) \to \text{Rect}(A) \rangle$$

- Derive $[\![\cdot]\!]'_{\text{pre}} : p \to (\Omega \underset{\text{pre}}{\rightsquigarrow}' B)$



In Theory...

Theorem (sound). $[\![\cdot]\!]'_{\mathrm{pre}}$ computes overapproximations of the preimages computed by $[\![\cdot]\!]_{\mathrm{pre}}$.

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

In Theory...

Theorem (sound). $[\![\cdot]\!]'_{\mathrm{pre}}$ computes overapproximations of the preimages computed by $[\![\cdot]\!]_{\mathrm{pre}}$.

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

Theorem (monotone). $[\![\cdot]\!]'_{\mathrm{pre}}$ is monotone.

- Consequence: Partitioning the domain never increases approximate preimages

In Theory...

Theorem (sound). $[\![\cdot]\!]'_{\mathrm{pre}}$ computes overapproximations of the preimages computed by $[\![\cdot]\!]_{\mathrm{pre}}$.

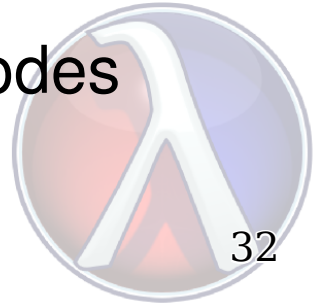
- Consequence: Sampling within preimages doesn't leave anything out

Theorem (monotone). $[\![\cdot]\!]'_{\mathrm{pre}}$ is monotone.

- Consequence: Partitioning the domain never increases approximate preimages

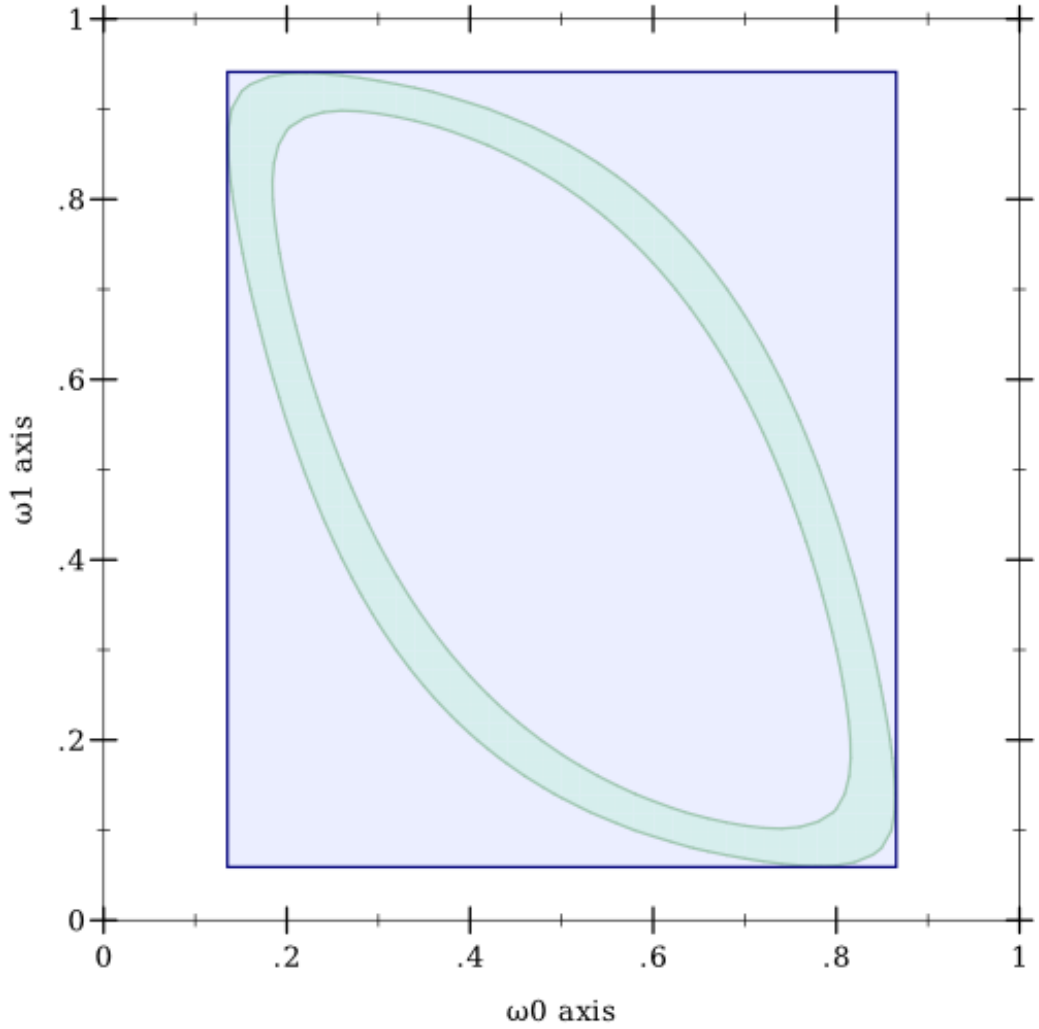
Theorem (decreasing). $[\![\cdot]\!]'_{\mathrm{pre}}$ never returns preimages larger than the given subdomain.

- Consequence: Refining preimage partitions never explodes



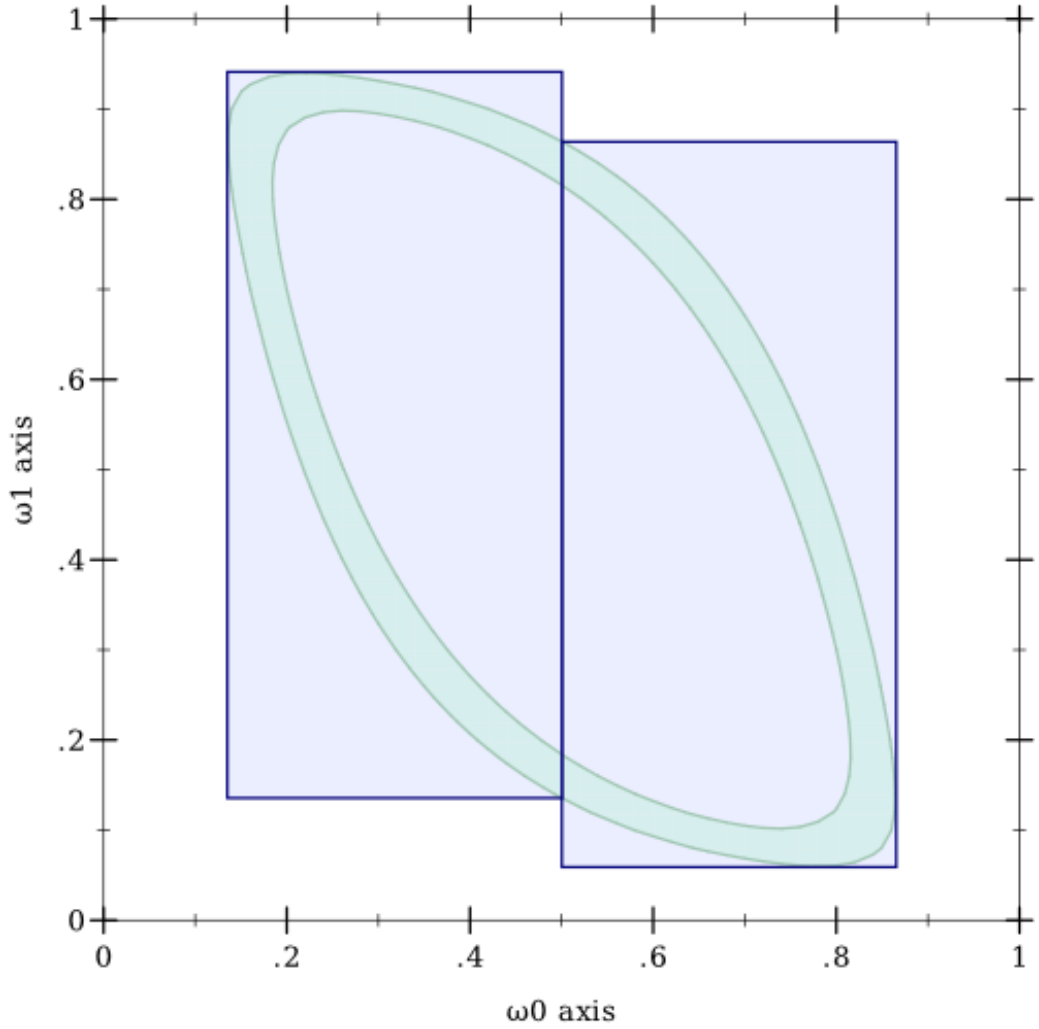
In Practice...

Theorems prove this always works:



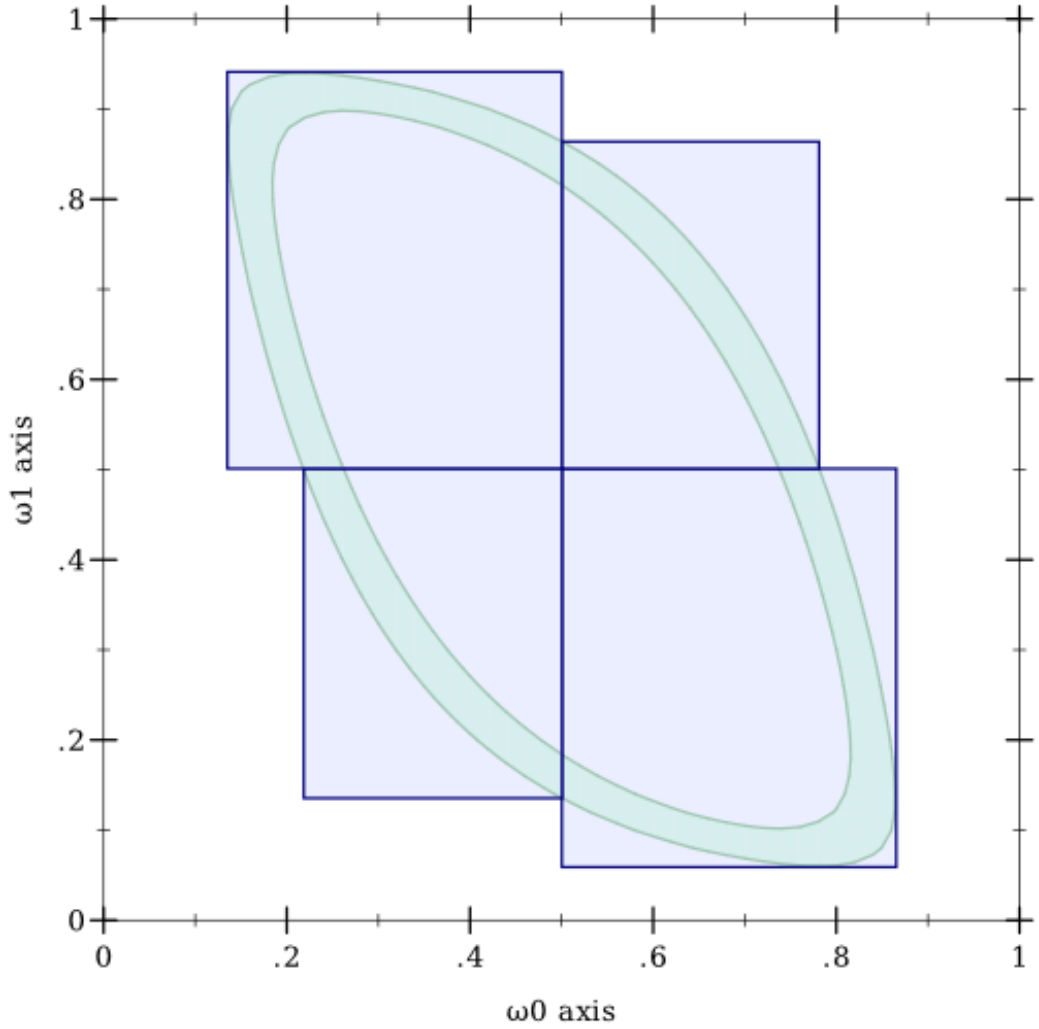
In Practice...

Theorems prove this always works:



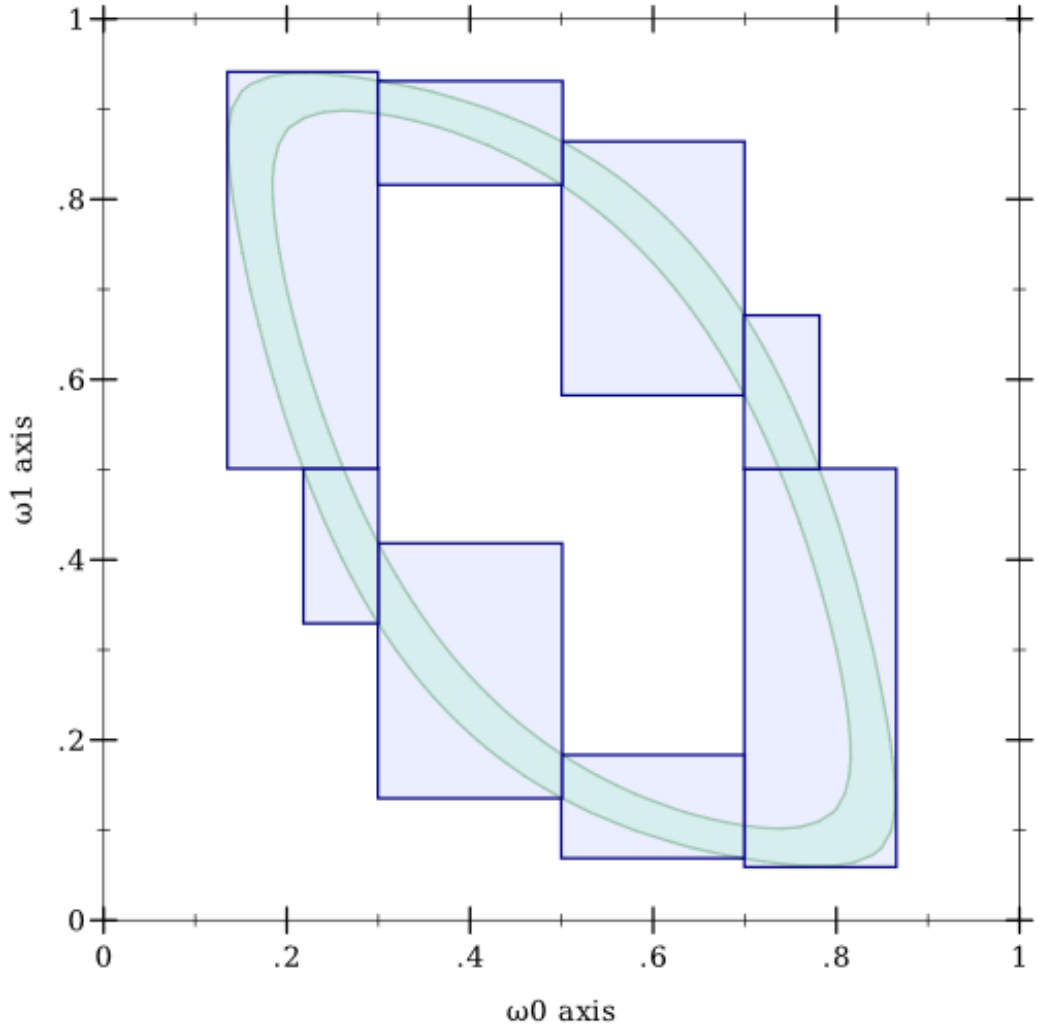
In Practice...

Theorems prove this always works:



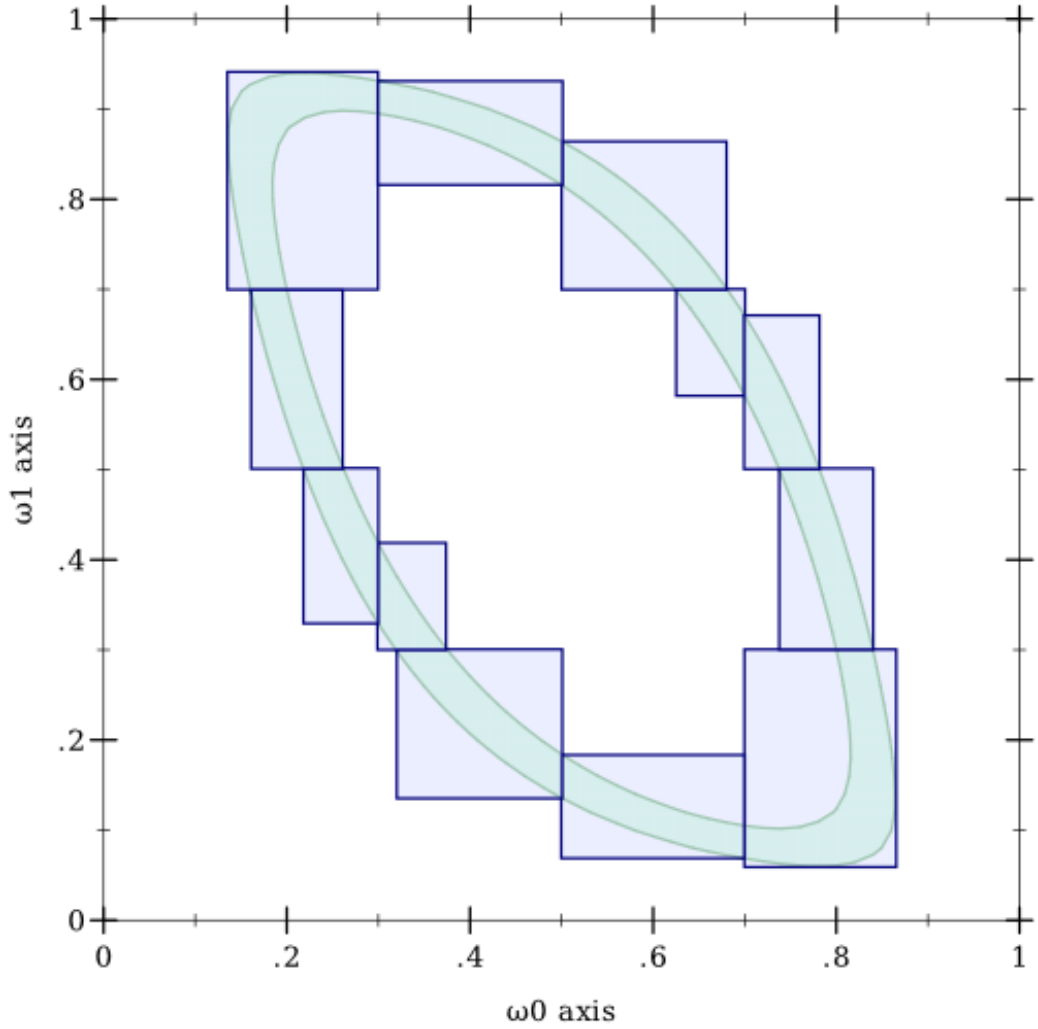
In Practice...

Theorems prove this always works:



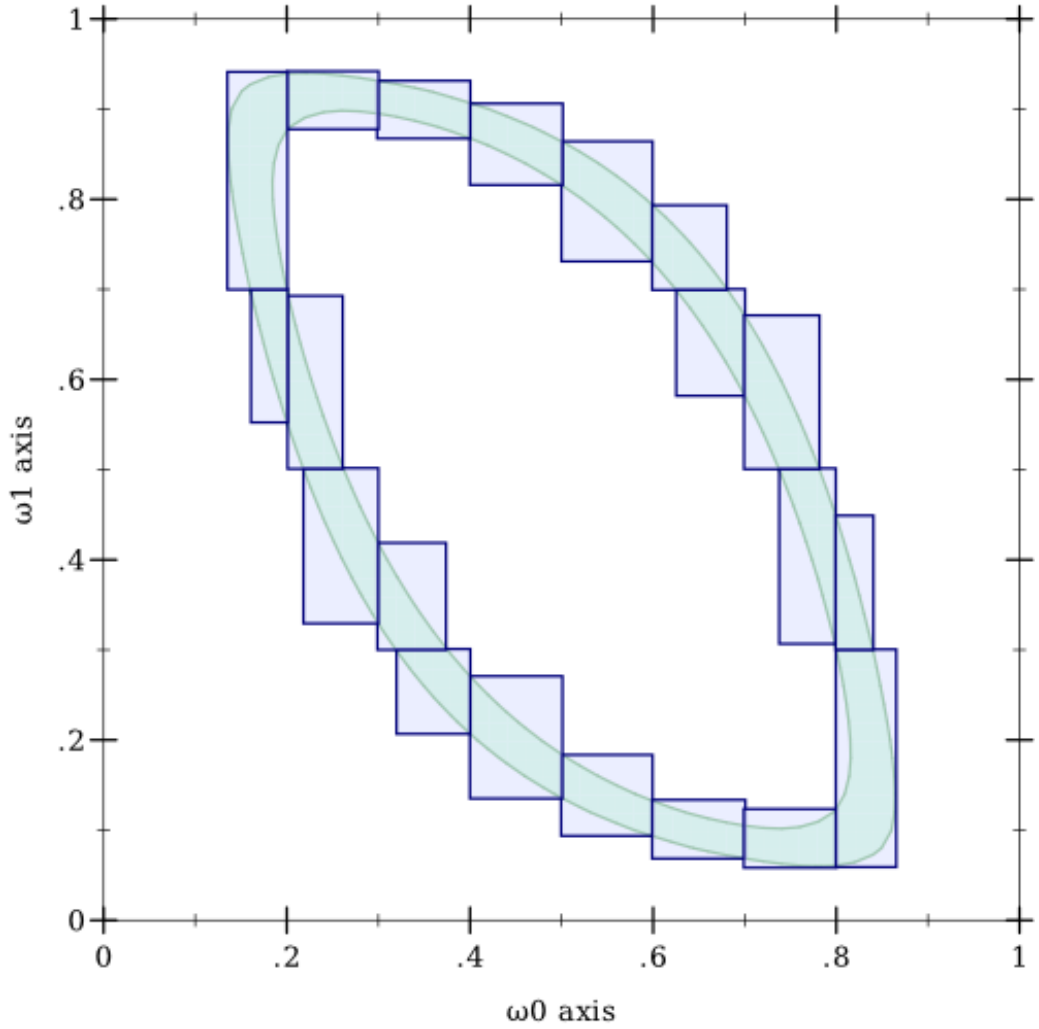
In Practice...

Theorems prove this always works:



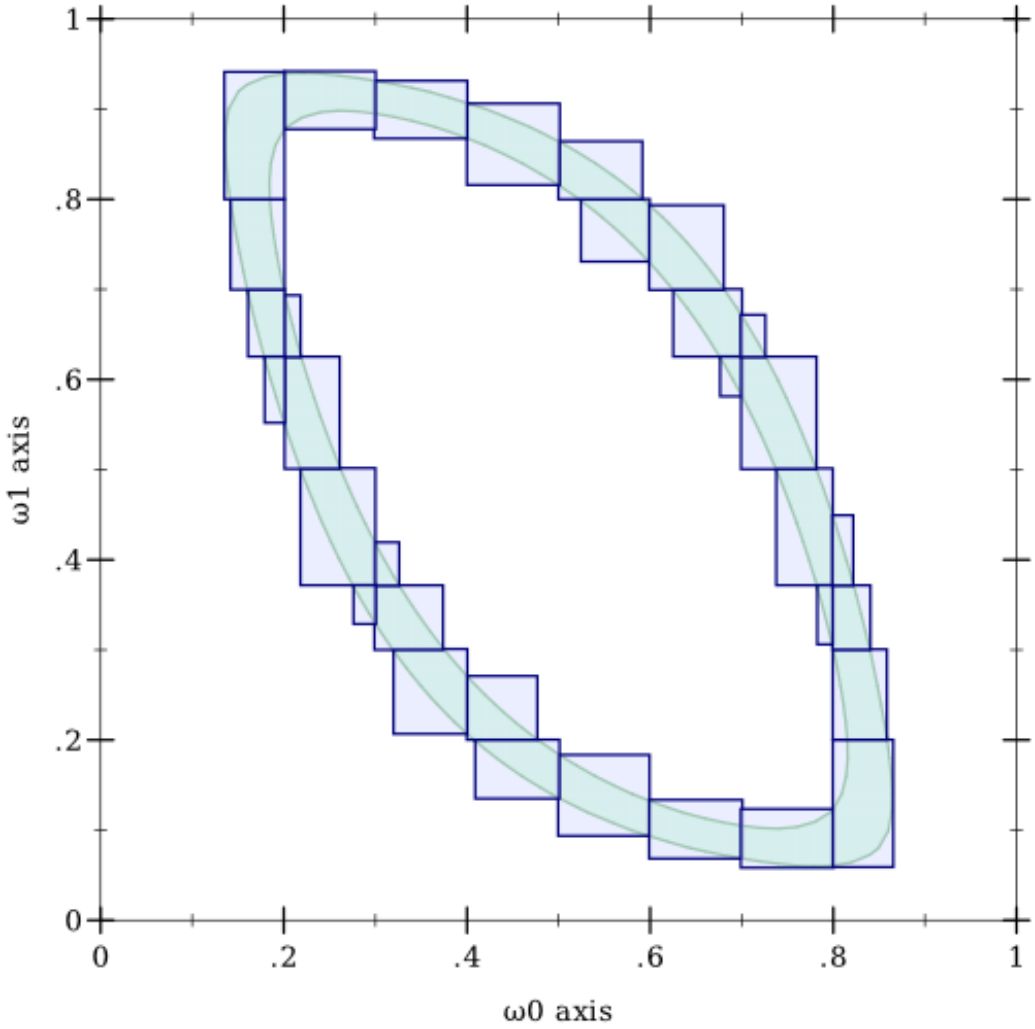
In Practice...

Theorems prove this always works:



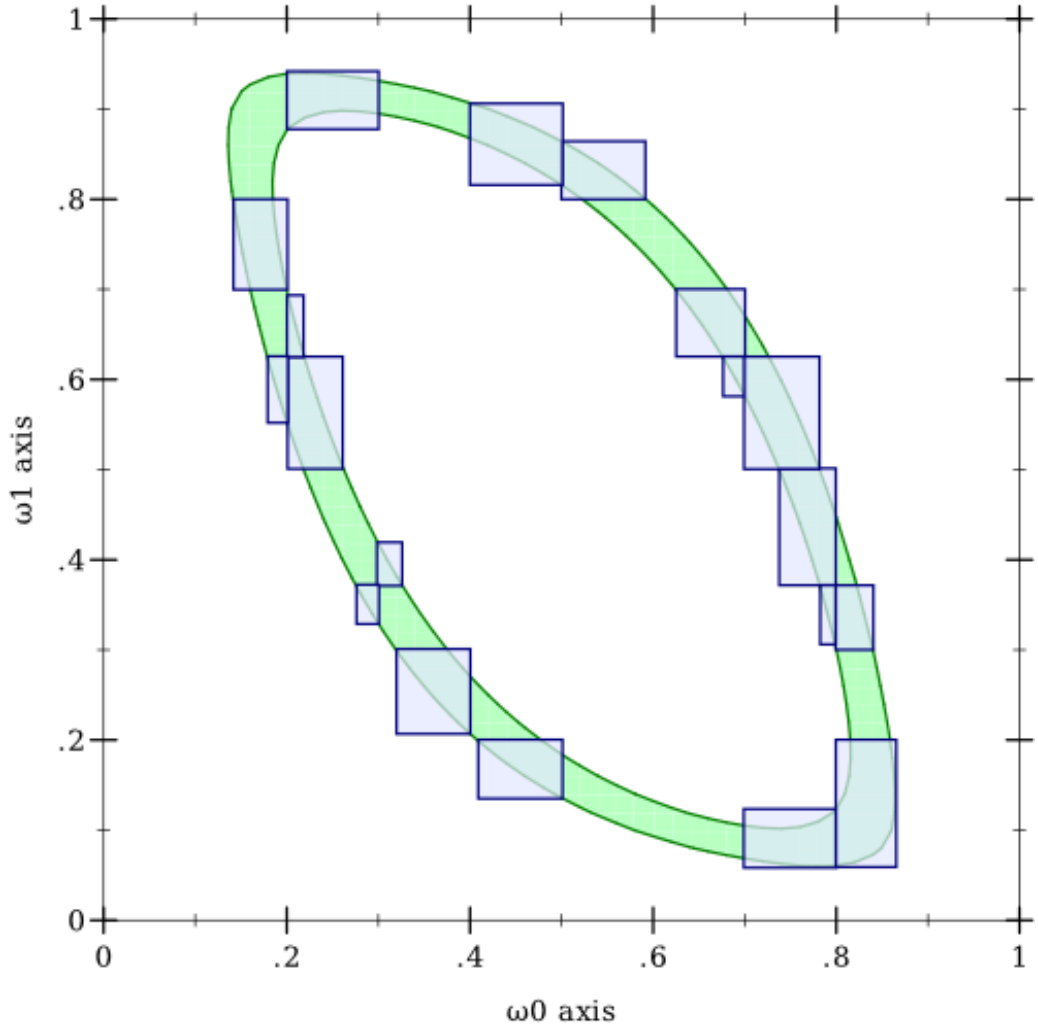
In Practice...

Theorems prove this always works:



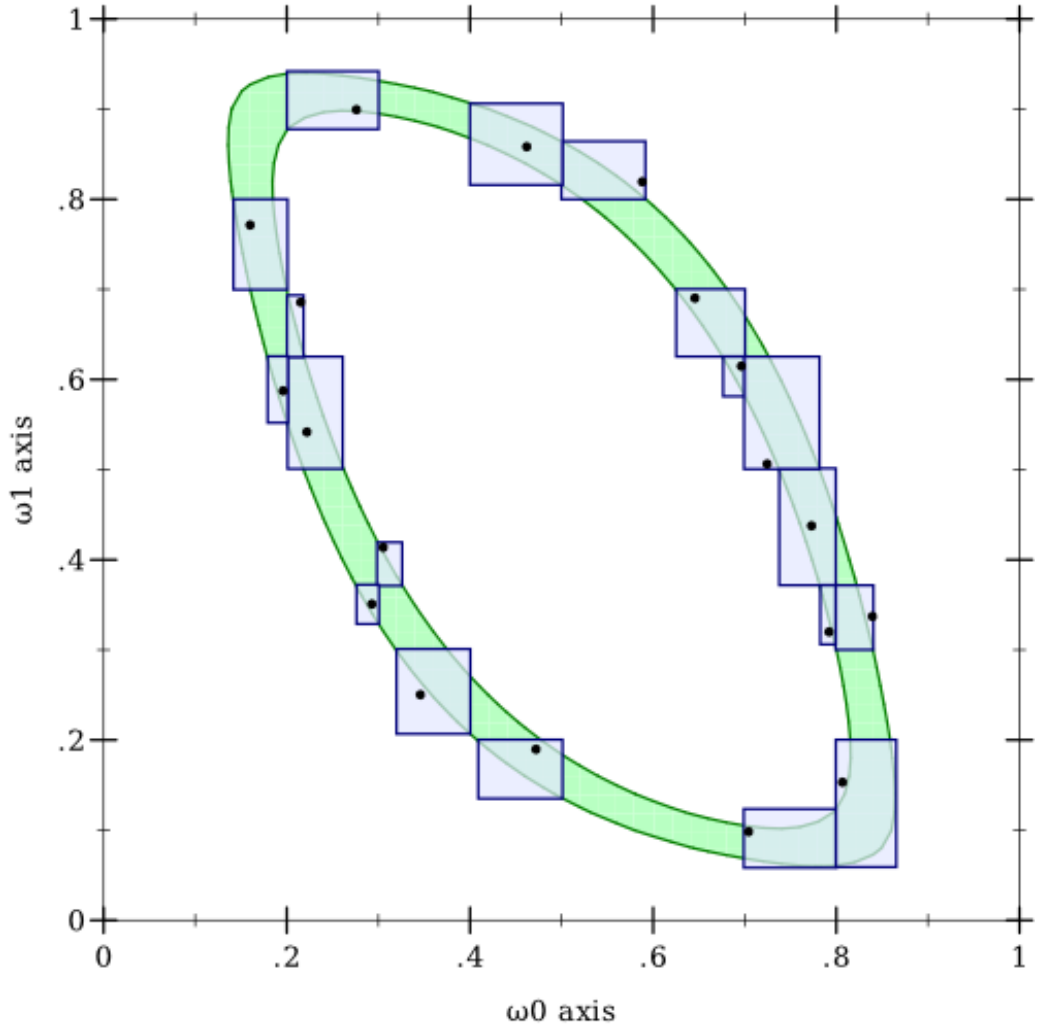
In Practice...

Theorems prove this always works:



In Practice...

Theorems prove this always works:



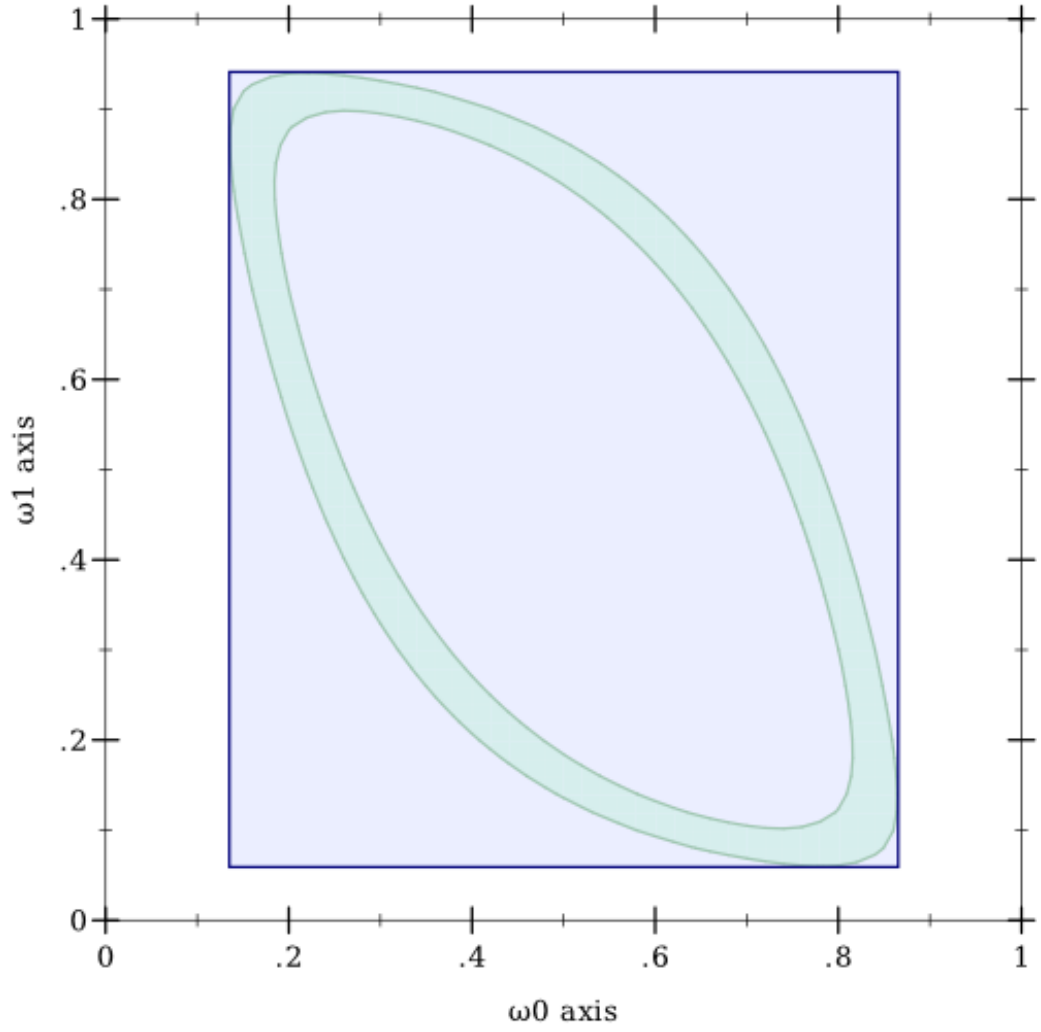
Importance Sampling

- Alternative to arbitrarily low-rate rejection sampling:

Importance Sampling

- Alternative to arbitrarily low-rate rejection sampling:

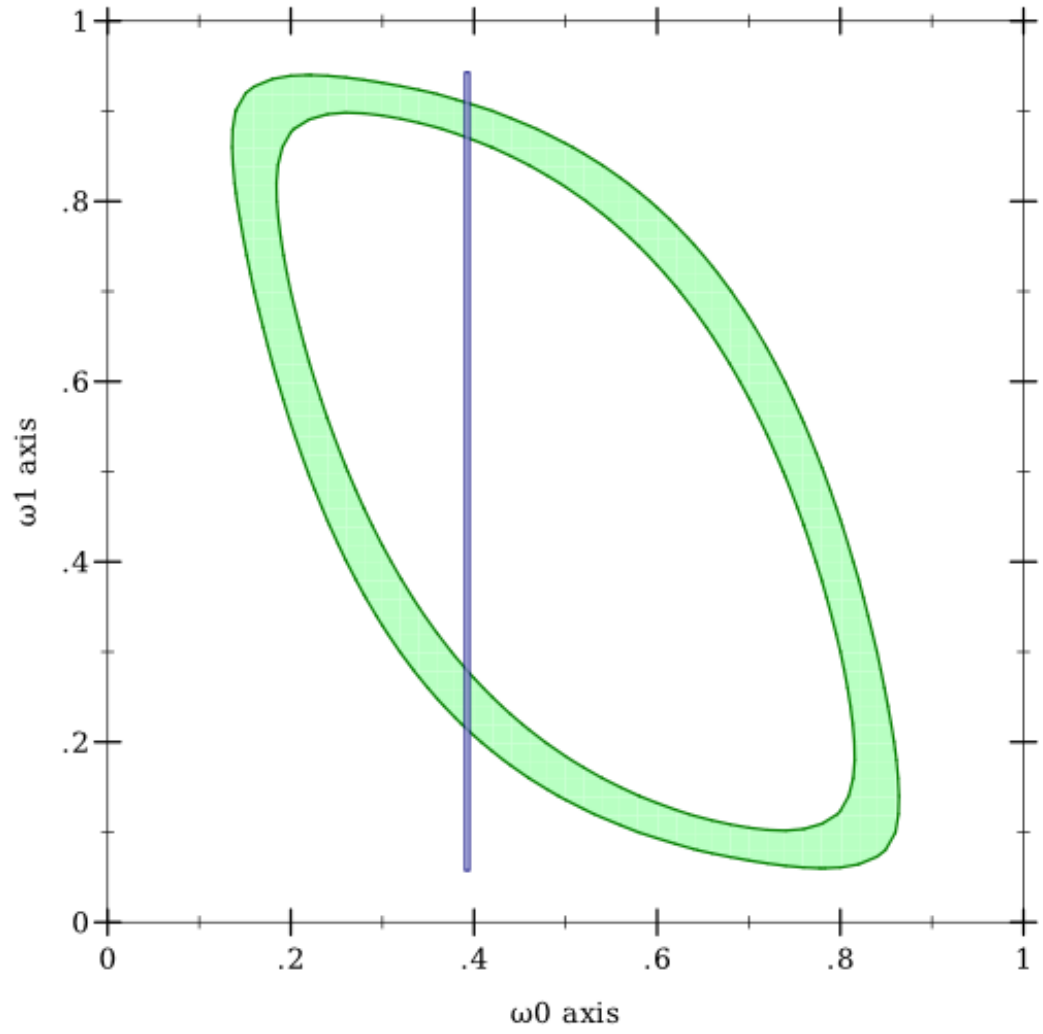
First, refine using preimage computation:



Importance Sampling

- Alternative to arbitrarily low-rate rejection sampling:

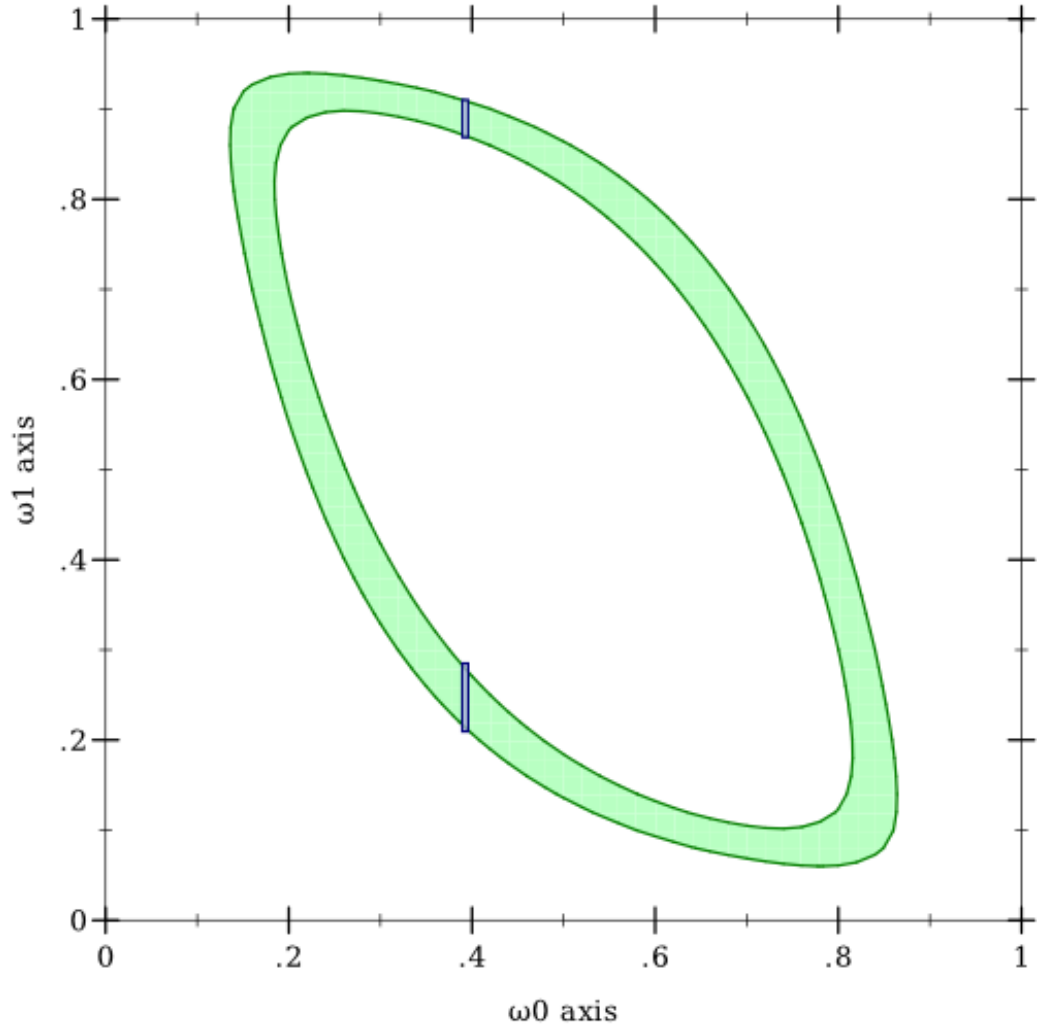
Second, randomly choose from arbitrarily fine partition:



Importance Sampling

- Alternative to arbitrarily low-rate rejection sampling:

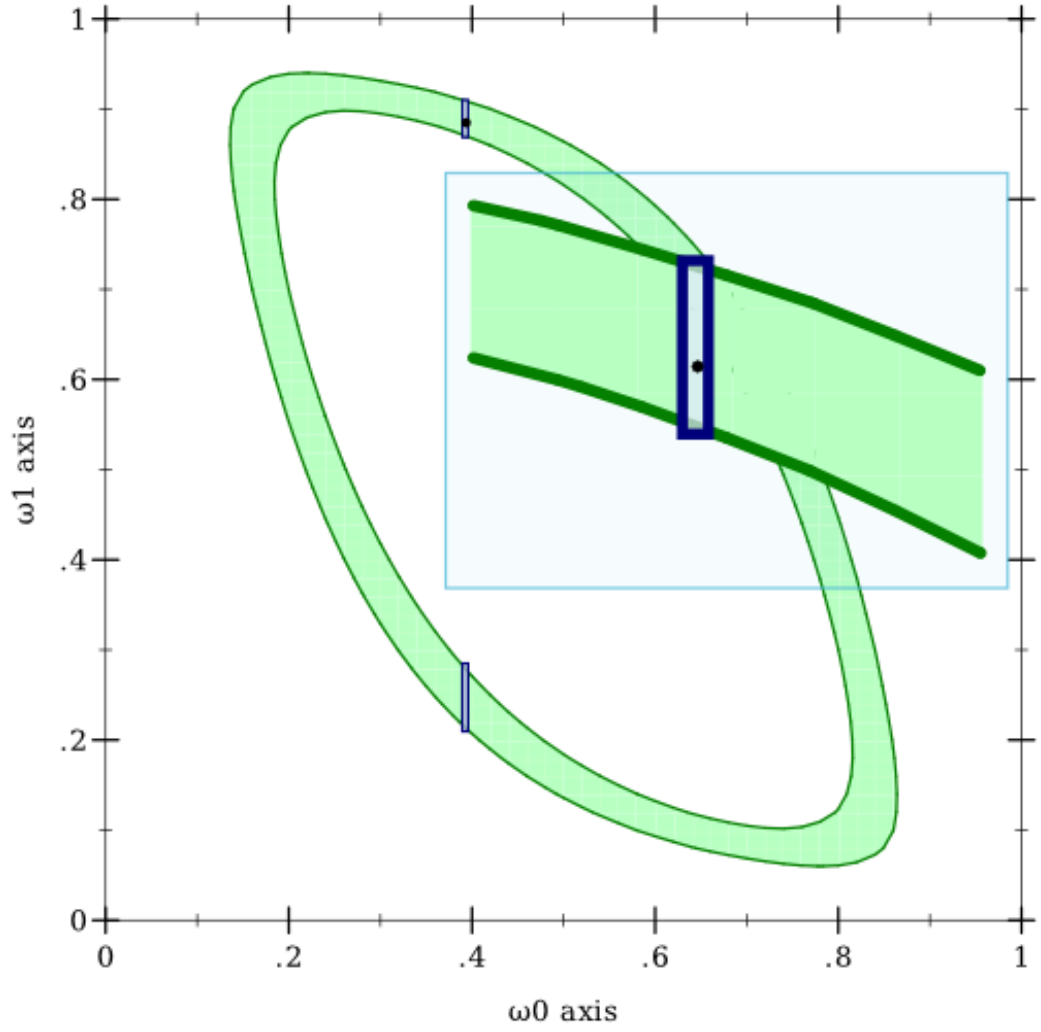
Third, refine again:



Importance Sampling

- Alternative to arbitrarily low-rate rejection sampling:

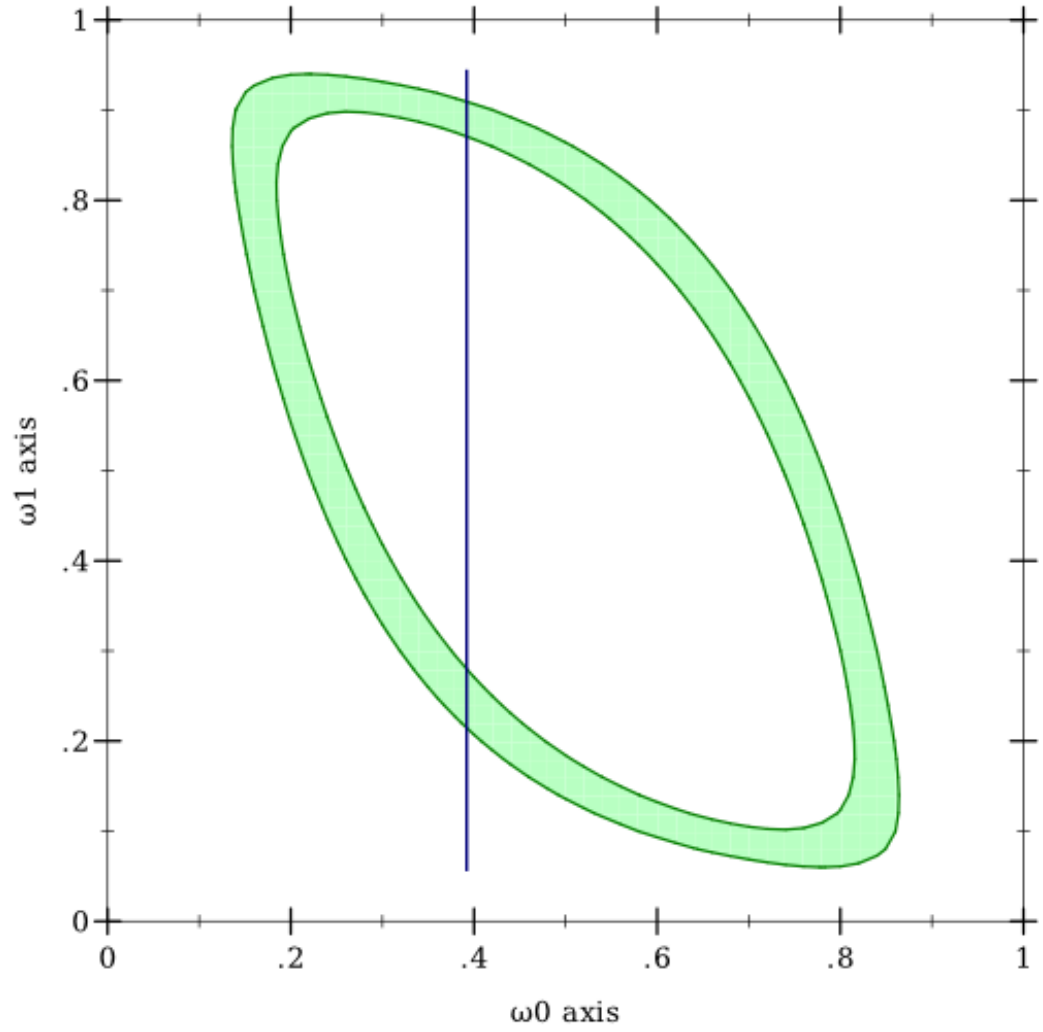
Fourth, sample uniformly:



Importance Sampling

- Alternative to arbitrarily low-rate rejection sampling:

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



Floating-Point Sensitivity and Accuracy

- Importance sampling is *extremely* sensitive to floating-point error

Floating-Point Sensitivity and Accuracy

- Importance sampling is *extremely* sensitive to floating-point error
- Requires either perfectly outwardly rounded interval arithmetic for monotone functions:

```
> (ivl (exp- 1.1) (exp+ 1.1))
(ivl 3.004166023946433 3.0041660239464334)
> (flonums-between 3.004166023946433 3.0041660239464334)
1
```

Floating-Point Sensitivity and Accuracy

- Importance sampling is *extremely* sensitive to floating-point error
- Requires either perfectly outwardly rounded interval arithmetic for monotone functions:

```
> (ivl (exp- 1.1) (exp+ 1.1))
(ivl 3.004166023946433 3.004166023946434)
> (flonums-between 3.004166023946433 3.004166023946434)
1
```

- Or a bound on its implementations' error to overapproximate upward- and downward-rounded implementations:

```
> (ivl (norm-inv-cdf- 0.8) (norm-inv-cdf+ 0.8))
(ivl 0.8416212335729142 0.8416212335729149)
> (flonums-between 0.8416212335729142 0.8416212335729149)
8
```

What About Infinite-Dimensional Models?

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

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

What About Infinite-Dimensional Models?

- 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 infinite and fully inlined:

What About Infinite-Dimensional Models?

- 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 infinite and fully inlined:

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

What About Infinite-Dimensional Models?

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

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

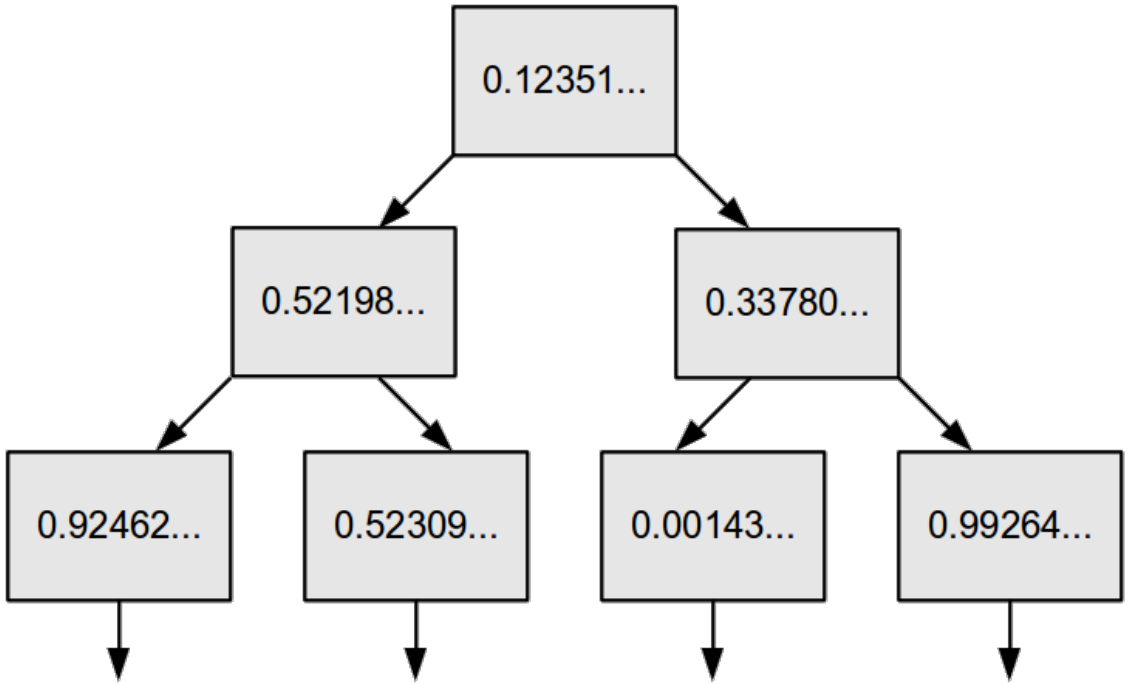
- Consider programs as being infinite and fully inlined:

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

- Random domain: an infinite tree of uniform random numbers

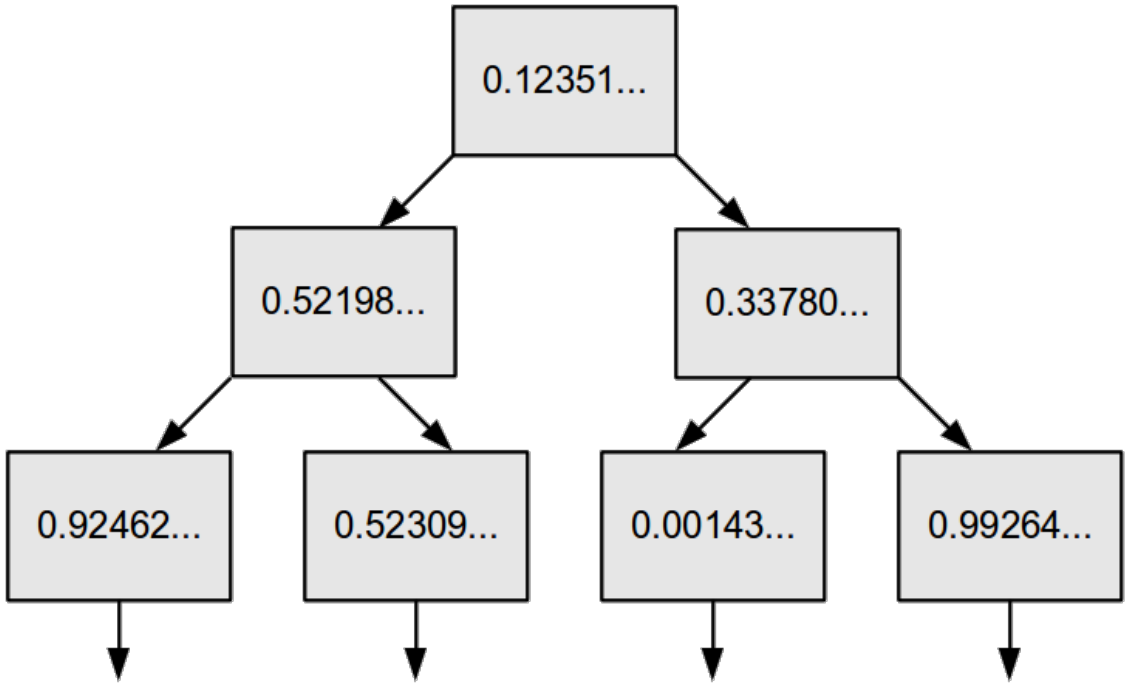
Domain Values

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



Domain Values

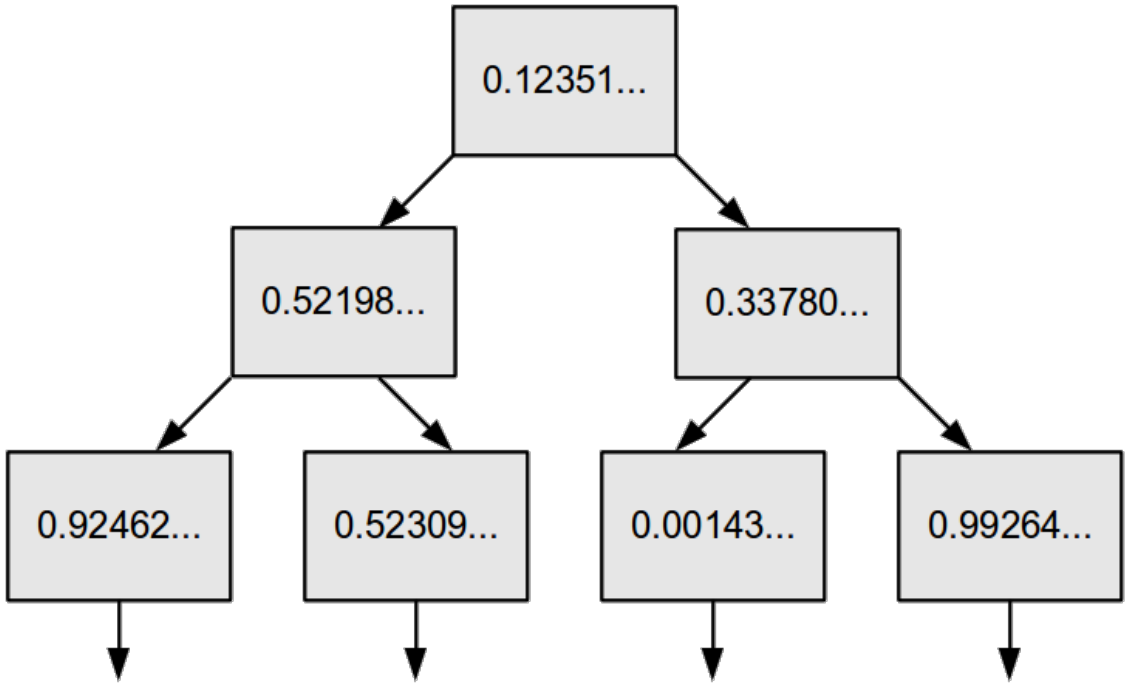
- Values $\omega \in \Omega$ are infinite binary trees:



- Implemented using lazy trees of random values

Domain Values

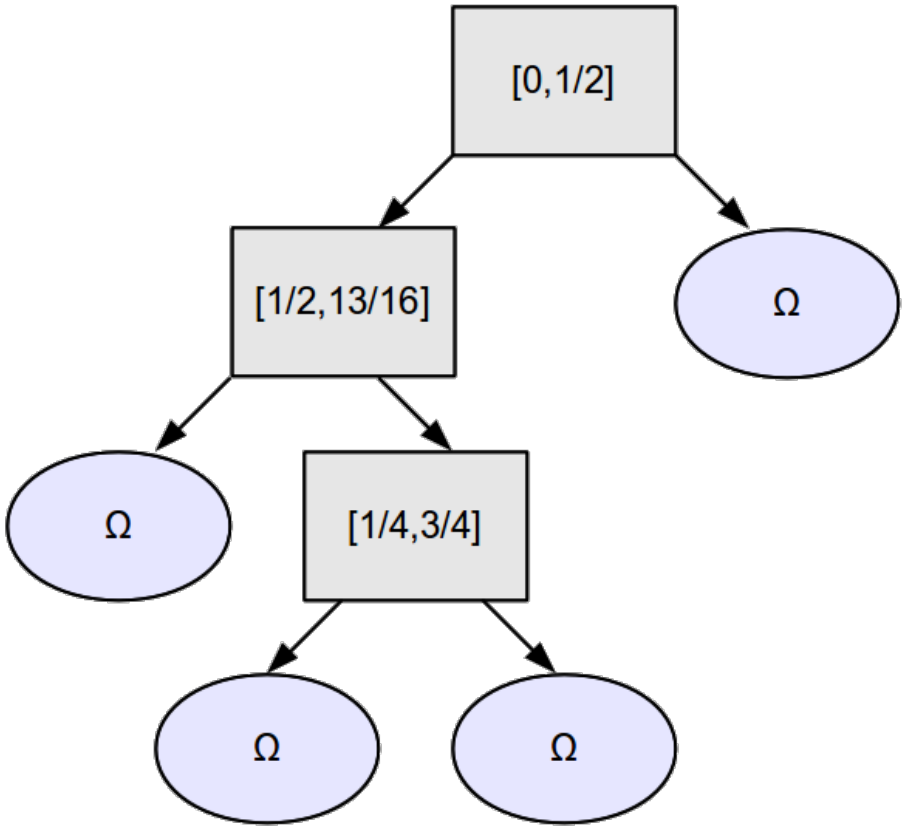
- Values $\omega \in \Omega$ are infinite binary trees:



- Implemented using lazy trees of random values
- No probability *density* for this, but there is a *measure*

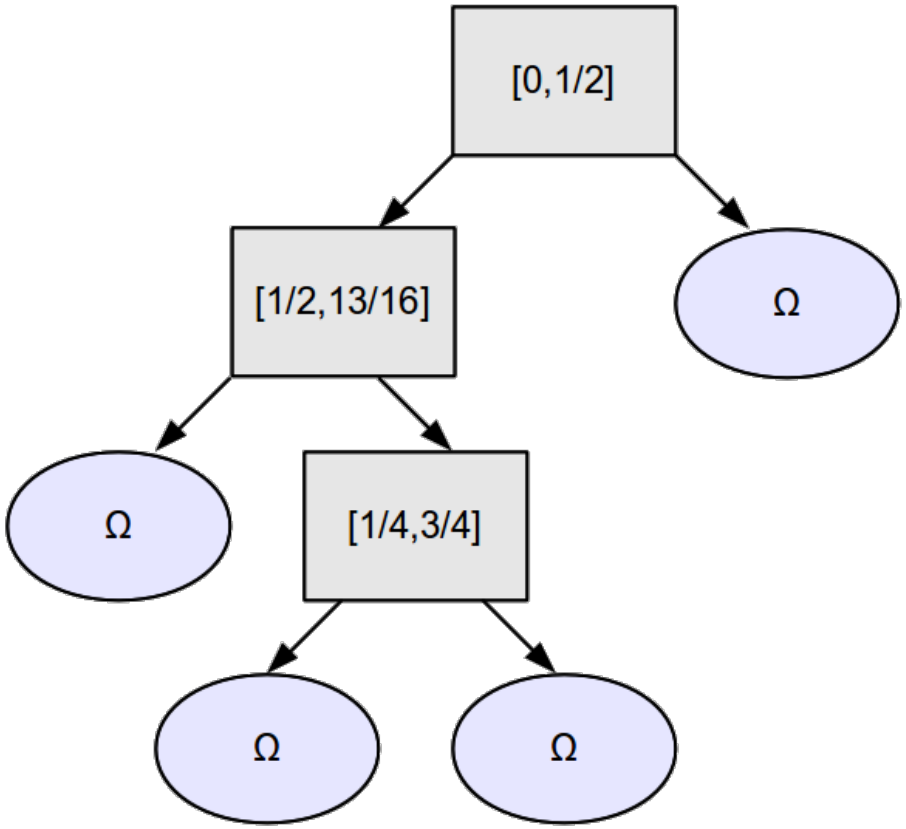
Domain Rectangles

- Rectangles $\Omega' \subseteq \Omega$ are products of tree axes:



Domain Rectangles

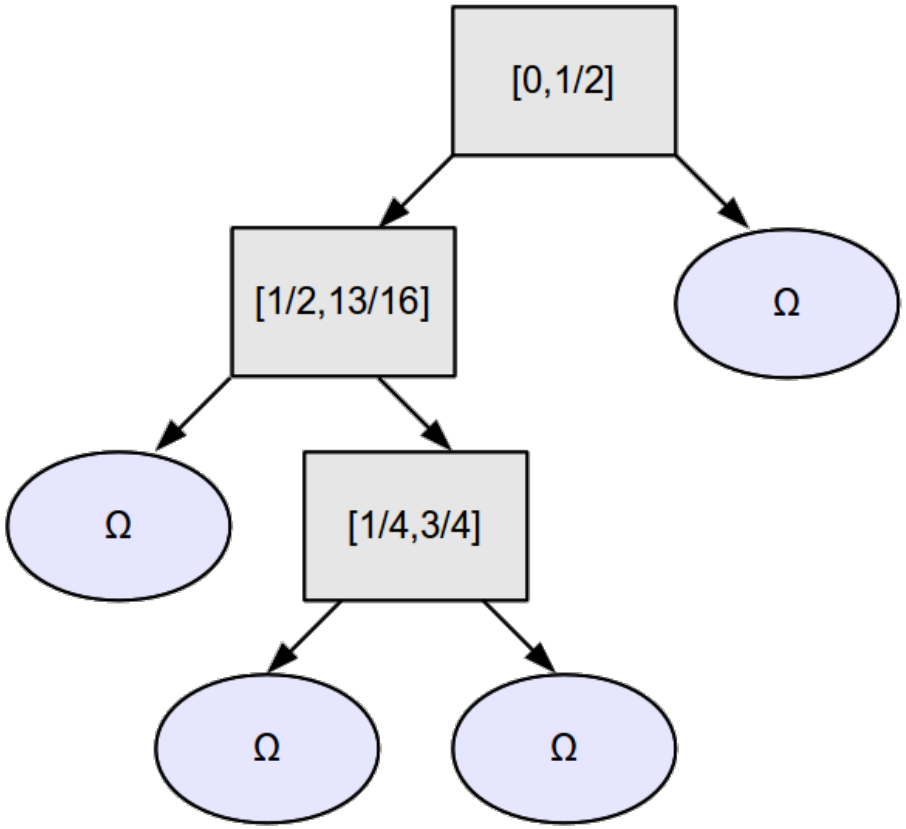
- Rectangles $\Omega' \subseteq \Omega$ are products of tree axes:



- Only finitely many axes are not $[0, 1]$

Domain Rectangles

- Rectangles $\Omega' \subseteq \Omega$ are products of tree axes:



- Only finitely many axes are not $[0, 1]$
- Implemented using finite trees of real sets

Recap: Problem and Solution

- Problem: Bayesian inference is hard

Recap: Problem and Solution

- Problem: Bayesian inference is hard
- Problem: Making a language for it is complicated by lack of density functions for programs with:

Recap: Problem and Solution

- Problem: Bayesian inference is hard
- Problem: Making a language for it is complicated by lack of density functions for programs with:
  - Discontinuous functions

Recap: Problem and Solution

- Problem: Bayesian inference is hard
- Problem: Making a language for it is complicated by lack of density functions for programs with:
  - Discontinuous functions
  - Unrestricted probabilistic conditions

Recap: Problem and Solution

- Problem: Bayesian inference is hard
- Problem: Making a language for it is complicated by lack of density functions for programs with:
  - Discontinuous functions
  - Unrestricted probabilistic conditions
  - Recursion and unbounded loops

Recap: Problem and Solution

- Problem: Bayesian inference is hard
- Problem: Making a language for it is complicated by lack of density functions for programs with:
  - Discontinuous functions
  - Unrestricted probabilistic conditions
  - Recursion and unbounded loops
- Solution: Language with measure-theoretic semantics

Solution Summary (1)

- Standard interpretation $[\![\cdot]\!]$
  - Directly implementable (with floating-point approximation)

Solution Summary (1)

- Standard interpretation $[\![\cdot]\!]$
  - Directly implementable (with floating-point approximation)
- Nonstandard interpretation $[\![\cdot]\!]_{\mathrm{pre}}$ for computing preimages
  - Unimplementable (except in λ_{ZFC})

Solution Summary (1)

- Standard interpretation $[\![\cdot]\!]$
  - Directly implementable (with floating-point approximation)
- Nonstandard interpretation $[\![\cdot]\!]_{\mathrm{pre}}$ for computing preimages
  - Unimplementable (except in λ_{ZFC})
  - Proof of correctness w.r.t. standard interpretation

Solution Summary (1)

- Standard interpretation $[\![\cdot]\!]$
  - Directly implementable (with floating-point approximation)
- Nonstandard interpretation $[\![\cdot]\!]_{\mathrm{pre}}$ for computing preimages
  - Unimplementable (except in λ_{ZFC})
  - Proof of correctness w.r.t. standard interpretation
- Approximating interpretation $[\![\cdot]\!]'_{\mathrm{pre}}$
  - Proofs of soundness w.r.t. nonstandard interpretation, and other nice properties

Solution Summary (1)

- Standard interpretation $[\![\cdot]\!]$
  - Directly implementable (with floating-point approximation)
- Nonstandard interpretation $[\![\cdot]\!]_{\mathrm{pre}}$ for computing preimages
  - Unimplementable (except in λ_{ZFC})
  - Proof of correctness w.r.t. standard interpretation
- Approximating interpretation $[\![\cdot]\!]'_{\mathrm{pre}}$
  - Proofs of soundness w.r.t. nonstandard interpretation, and other nice properties
  - Directly implementable, given a rectangular set library

Solution Summary (2)

- Rectangular set library
  - Sets required by semantics: rectangular subsets of Ω, $\mathbb{B}\mathrm{ool}$, finite cartesian products

Solution Summary (2)

- Rectangular set library
  - Sets required by semantics: rectangular subsets of Ω, Bool, finite cartesian products
  - Implementation-specific rectangles: $\{\mathrm{null}\}$, interval unions, tagged structures

Solution Summary (2)

- Rectangular set library
  - Sets required by semantics: rectangular subsets of Ω, Bool, finite cartesian products
  - Implementation-specific rectangles: $\{\mathrm{null}\}$, interval unions, tagged structures
  - Values for each rectangular set type

Solution Summary (2)

- Rectangular set library
  - Sets required by semantics: rectangular subsets of Ω, Bool, finite cartesian products
  - Implementation-specific rectangles: $\{\mathrm{null}\}$, interval unions, tagged structures
  - Values for each rectangular set type
- Two sampling algorithms (so far!)
  - Use interpretations to sample uniformly from preimages

Solution Summary (2)

- Rectangular set library
  - Sets required by semantics: rectangular subsets of Ω, Bool, finite cartesian products
  - Implementation-specific rectangles: $\{\mathrm{null}\}$, interval unions, tagged structures
  - Values for each rectangular set type
- Two sampling algorithms (so far!)
  - Use interpretations to sample uniformly from preimages
  - Correct for any program, any positive-probability condition (up to floating-point error)

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 $X, Y \mid \sqrt{X^2 + Y^2} = 1$
- Implementation:

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

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))))))
```

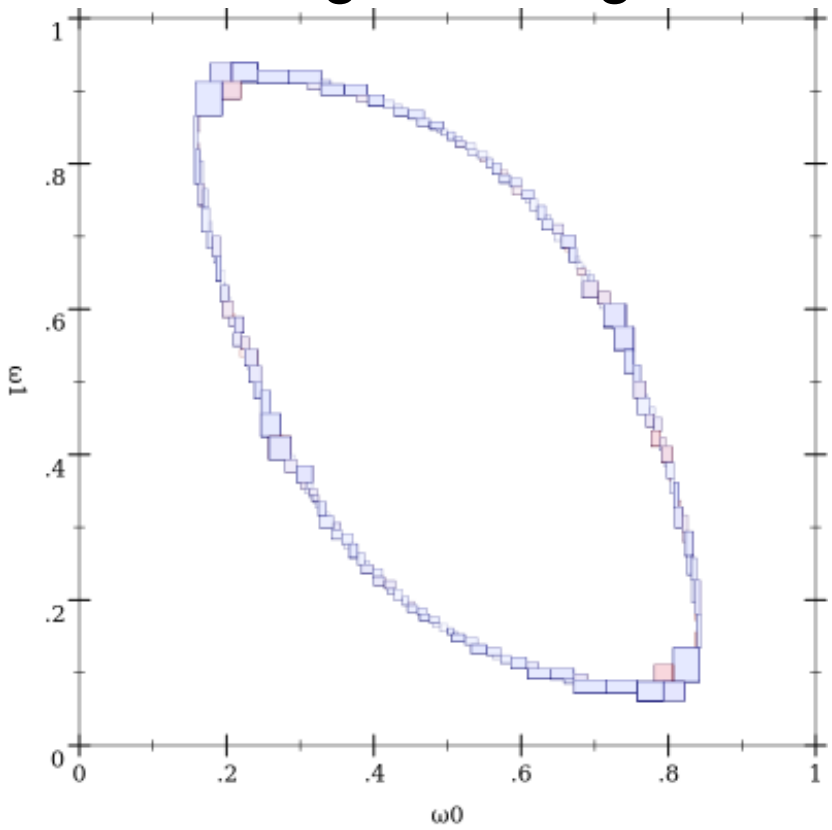
- Goal: Sample in the preimage of

```
(set-list reals reals (interval (- 1.0 ε) (+ 1.0 ε)))
```

Demo: Normal-Normal With Circular Condition

For ε = 0.01:

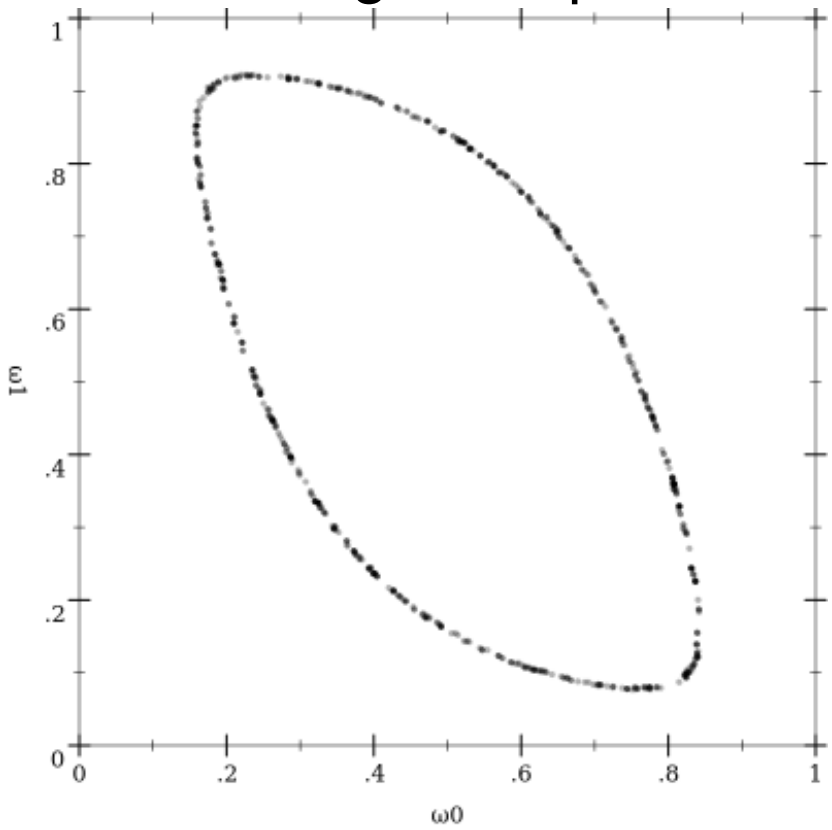
Preimage rectangles



Demo: Normal-Normal With Circular Condition

For $\varepsilon = 0.01$:

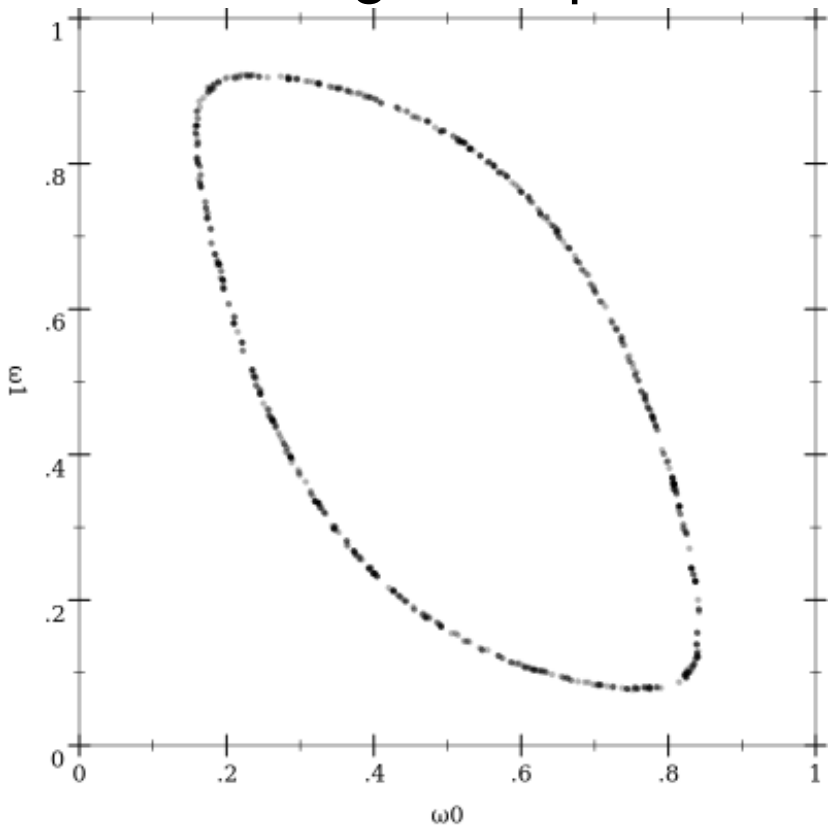
Preimage samples



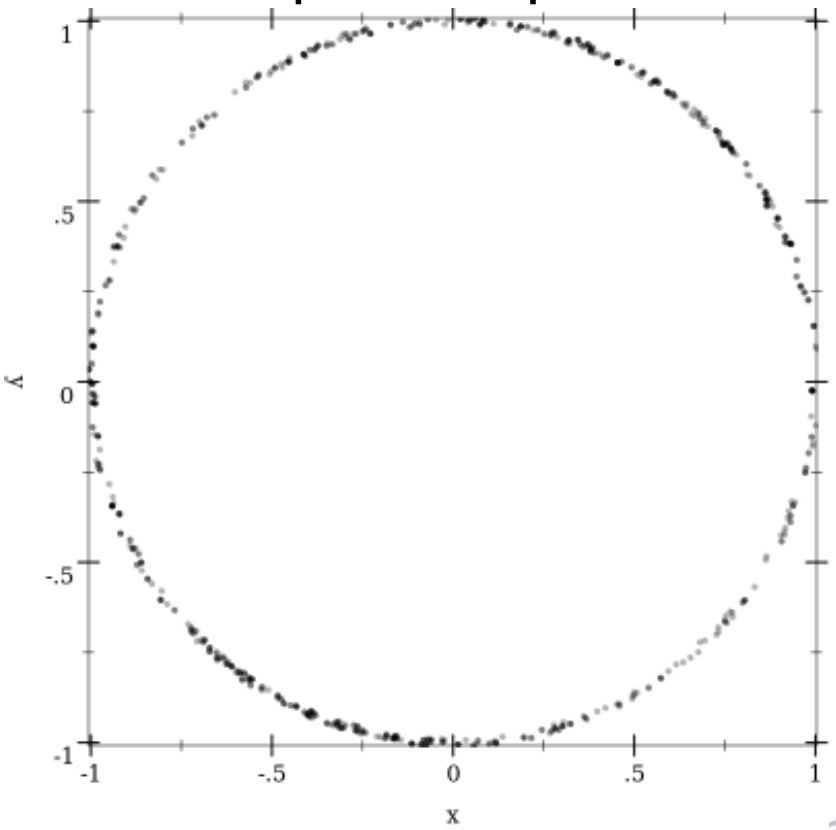
Demo: Normal-Normal With Circular Condition

For $\varepsilon = 0.01$:

Preimage samples



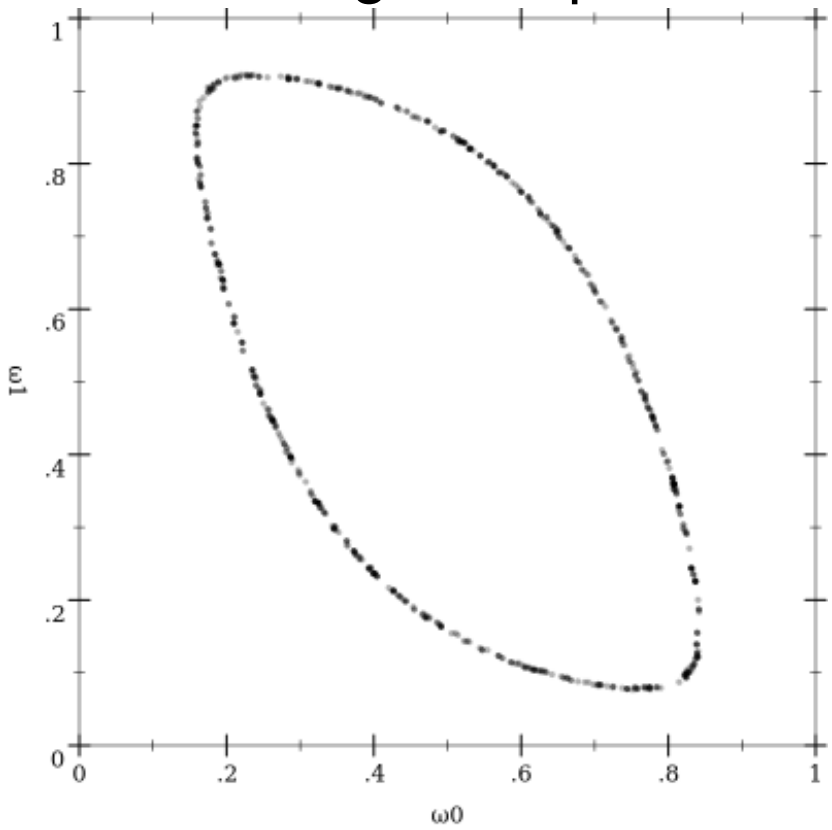
Output samples



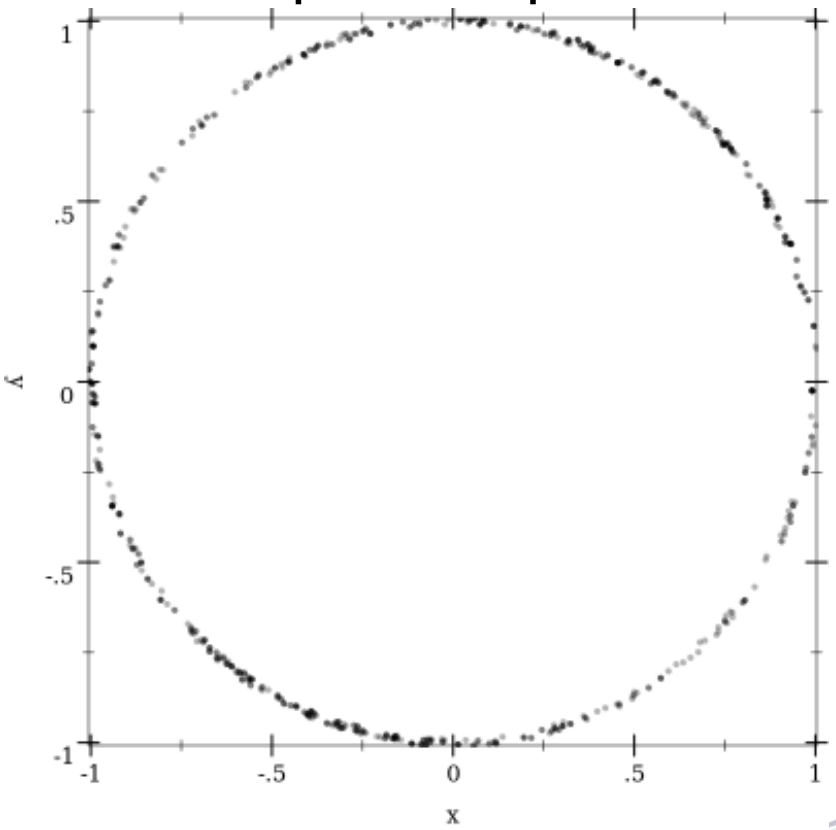
Demo: Normal-Normal With Circular Condition

For $\varepsilon = 0.01$:

Preimage samples



Output samples



- Works fine with much smaller ε

Demo: Thermometer

- Normal-Normal thermometer process:

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

Demo: Thermometer

- Normal-Normal thermometer process:

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

- Objective: Find the distribution of $X \mid Y' = 100$

Demo: Thermometer

- Normal-Normal thermometer process:

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

- 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))))
```

Demo: Thermometer

- Normal-Normal thermometer process:

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

- Objective: Find the distribution of $X \mid Y' = 100$
- Implementation:

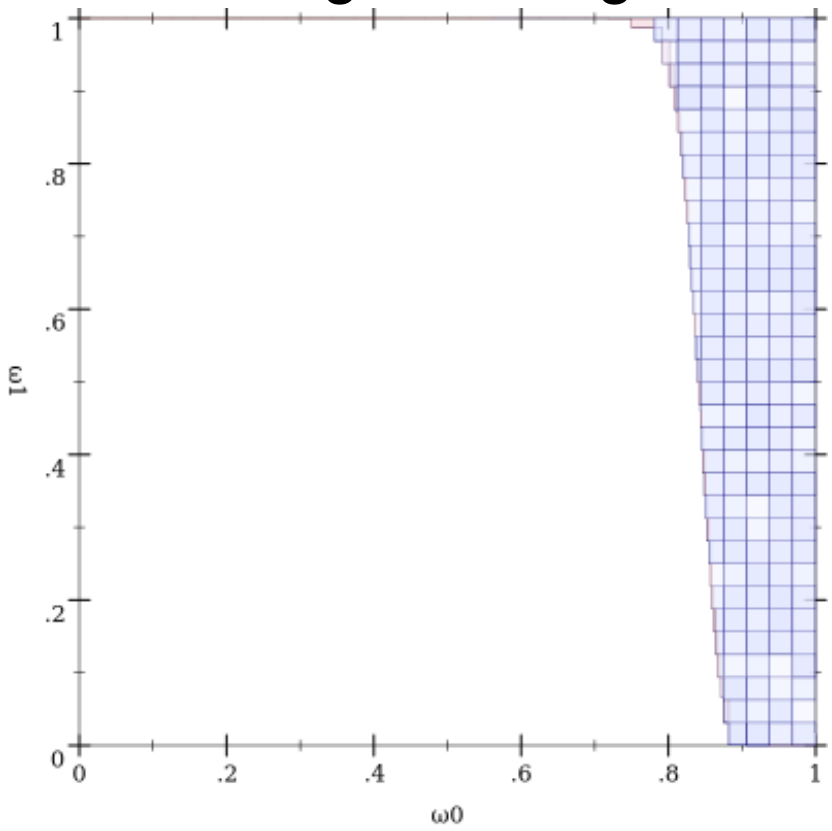
```
(define/drbayes e
  (let* ([x  (normal 90 10)]
         [y  (normal x 1)])
    (list x (if (> y 100) 100 y))))
```

- Goal: Sample in the preimage of

```
(set-list reals (interval 100.0 100.0))
```

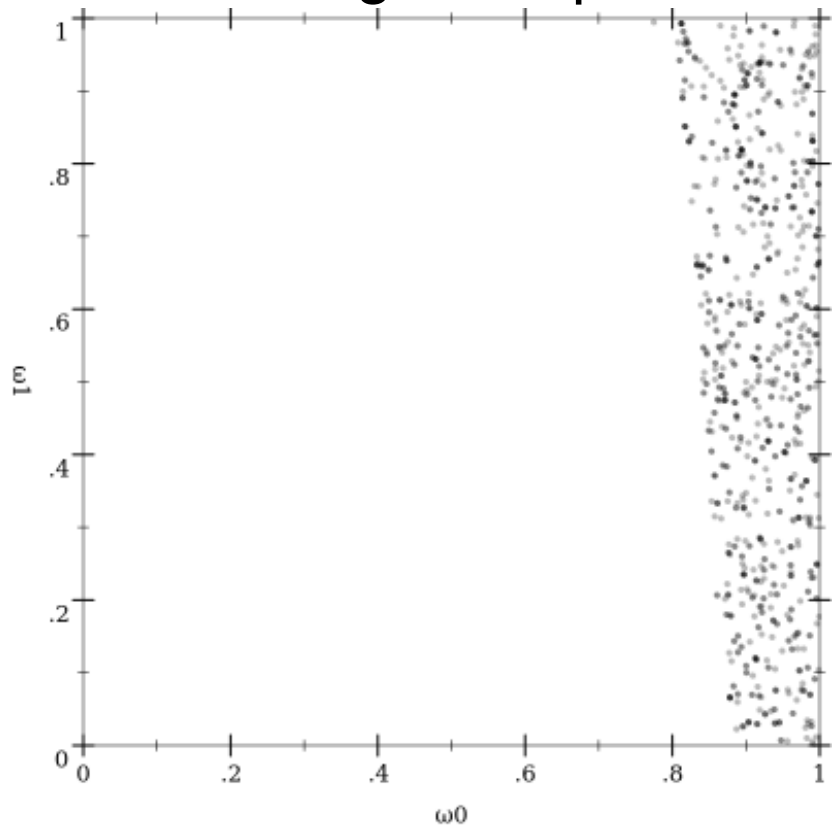
Demo: Thermometer

Preimage rectangles



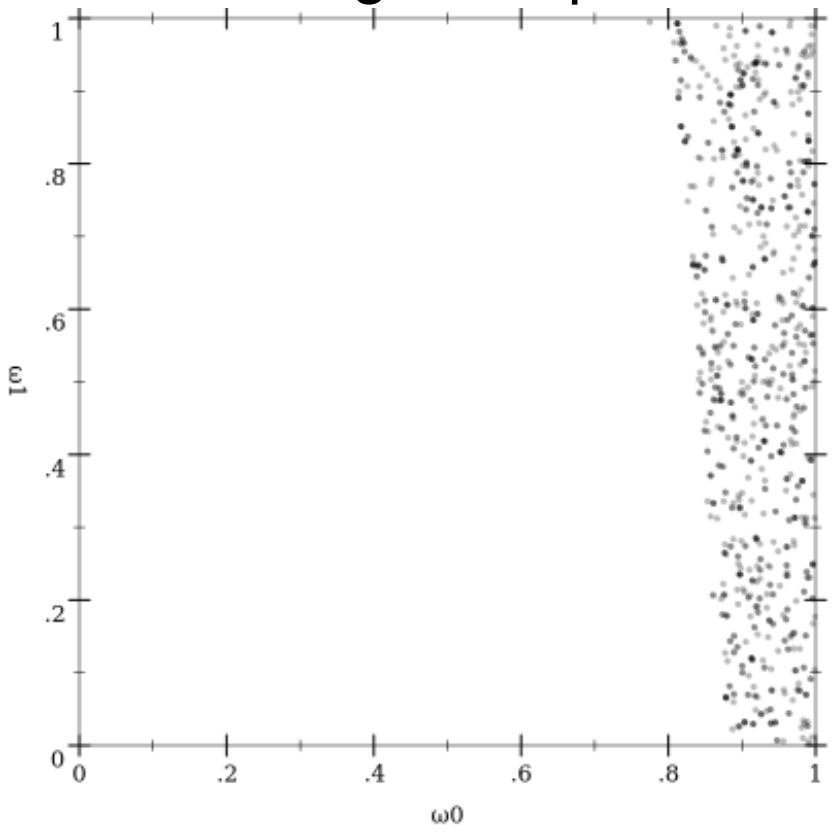
Demo: Thermometer

Preimage samples

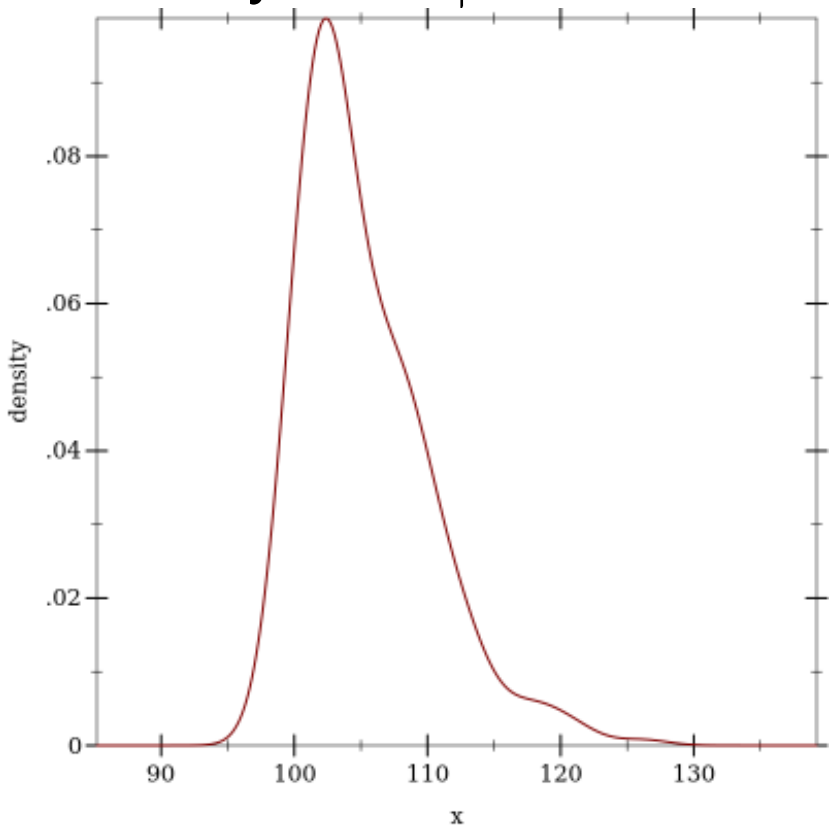


Demo: Thermometer

Preimage samples

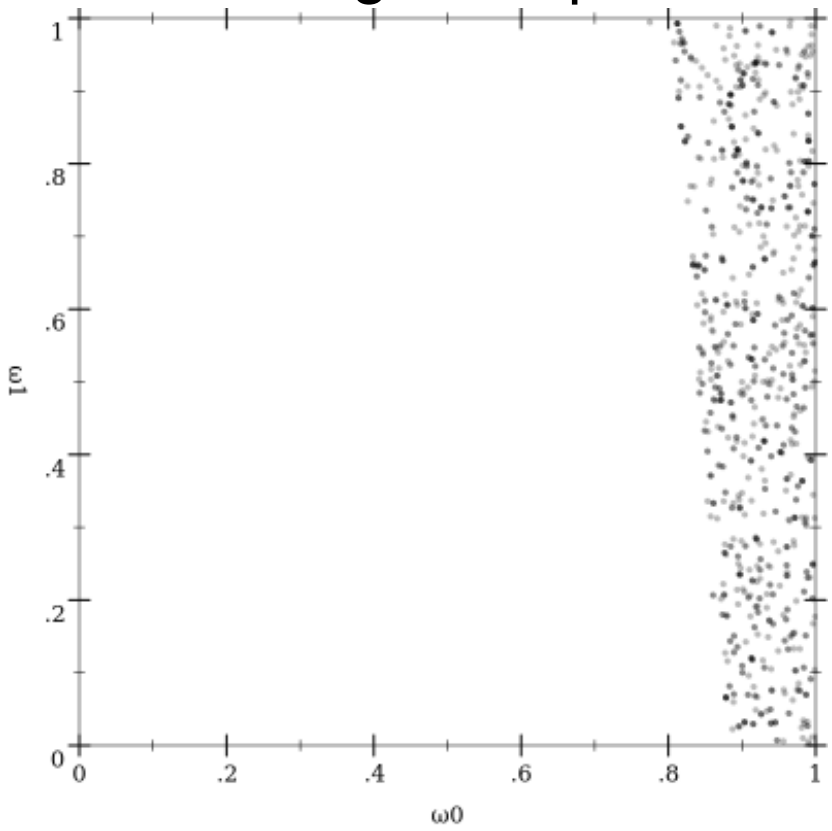


Density of $X \mid Y' = 100$

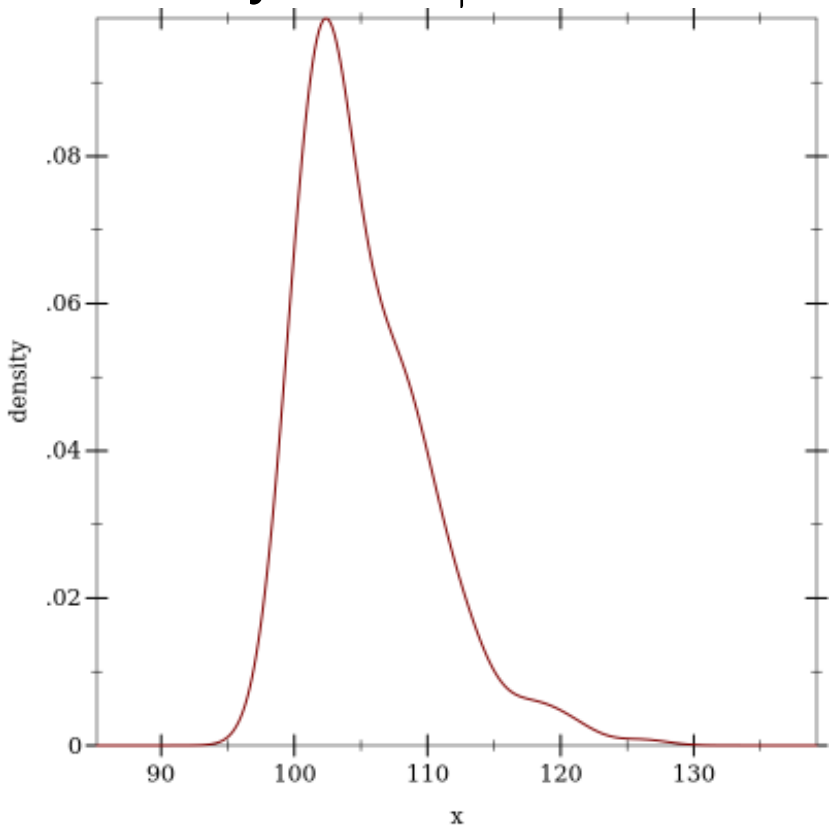


Demo: Thermometer

Preimage samples



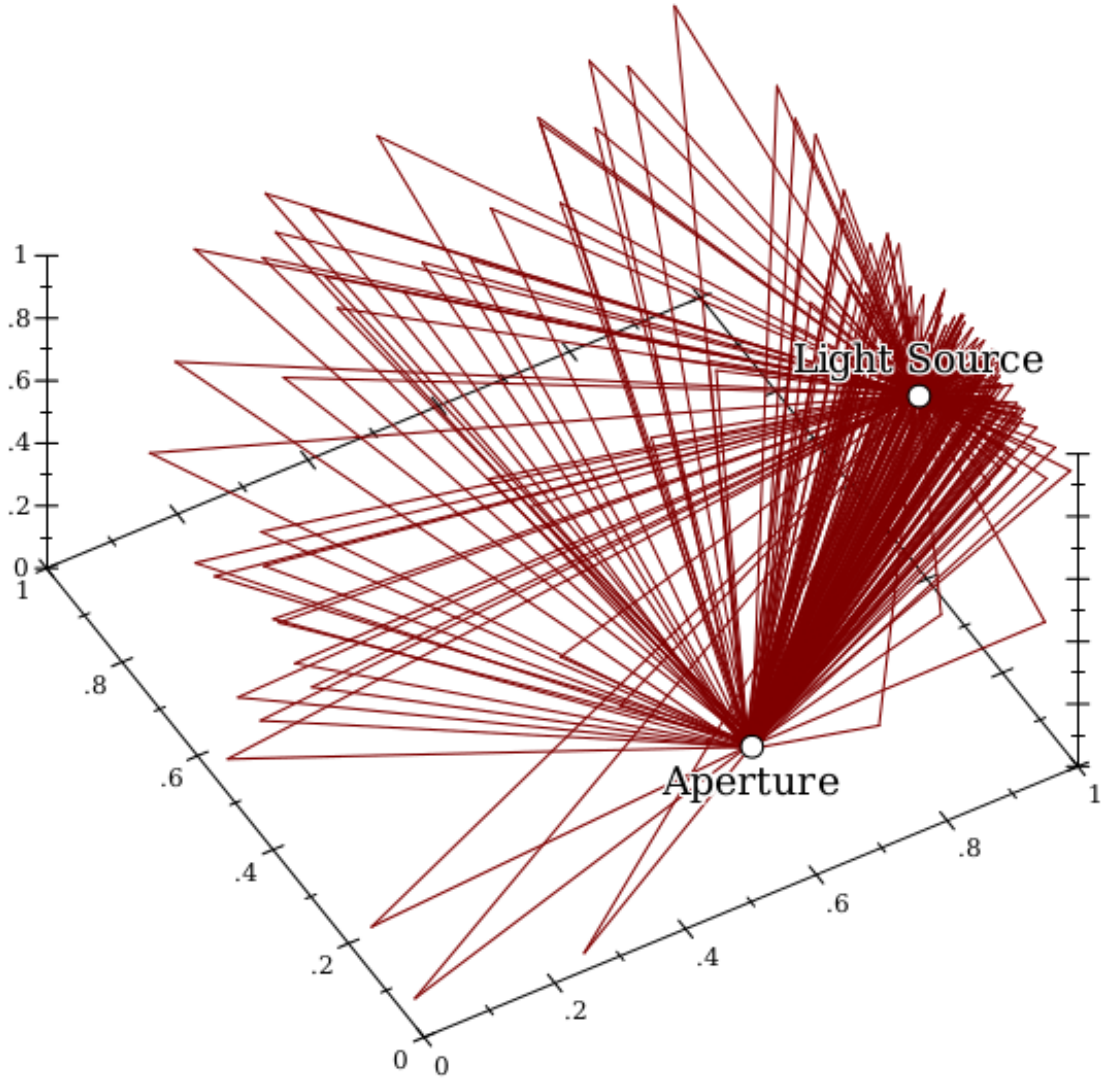
Density of $X \mid Y' = 100$



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

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

```
(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)))
```

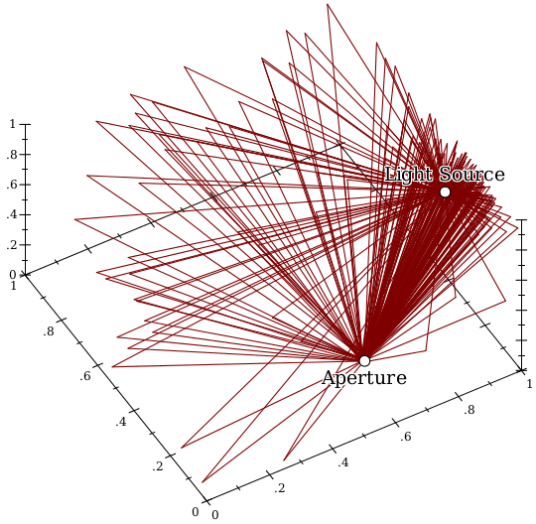
Demo: Stochastic Ray Tracing

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

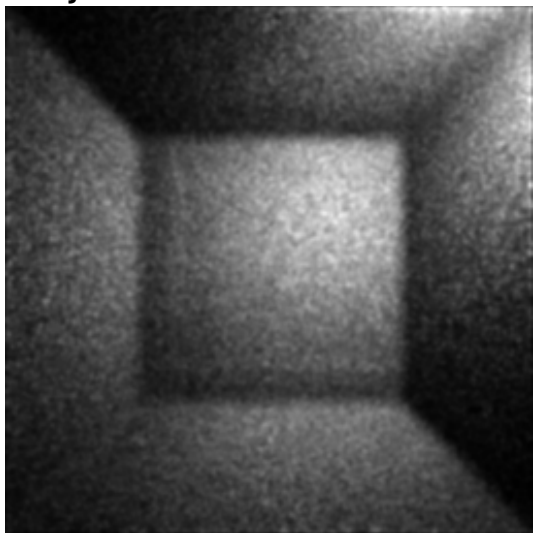
```
(define/drbayes (ray-plane-intersect p0 v n d)
  (let ([denom  (- (vec-dot v n))])
    (if (positive? denom)
        (let ([t  (/ (+ d (vec-dot p0 n)) denom)])
          (if (positive? t)
              (collision t (vec+ p0 (vec-scale v t)) n)
              #f))
        #f)))
```

- Constrained light path outputs:

Paths Through Aperture



Projected and Accumulated



Future Work

- Mostly efficiency/scalability

Future Work

- Mostly efficiency/scalability
- Incrementalization

Future Work

- Mostly efficiency/scalability
- Incrementalization
- Other set representations (e.g. convex polyhedra, regular expressions)

Future Work

- Mostly efficiency/scalability
- Incrementalization
- Other set representations (e.g. convex polyhedra, regular expressions)
- Type systems and other static analysis

Future Work

- Mostly efficiency/scalability
- Incrementalization
- Other set representations (e.g. convex polyhedra, regular expressions)
- Type systems and other static analysis
- More, smarter sampling algorithms

Future Work

- Mostly efficiency/scalability
- Incrementalization
- Other set representations (e.g. convex polyhedra, regular expressions)
- Type systems and other static analysis
- More, smarter sampling algorithms
- Debugging support

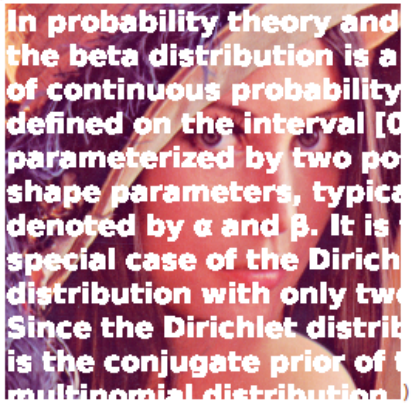
Future Work

Untitled 3 - DrRacket*

Untitled 3 (define ...) Run Stop

```
#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- σ)
  (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- σ))))

(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 (reconstruct img)
  ....)
```

```
Welcome to DrRacket, version 6.0.0.1--2013-12-12(c321f6dd/d) [3m].
Language: racket; memory limit: 1024 MB.
> (reconstruct
```



```
>
```

Determine language from source 7:2 639.69 MB

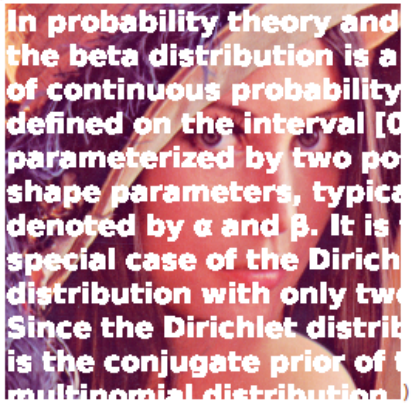
Future Work

Untitled 3 - DrRacket*

Untitled 3 (define ...) Run Stop

```
#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- σ)
  (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- σ))))

(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 (reconstruct img)
  ....)
```

```
Welcome to DrRacket, version 6.0.0.1--2013-12-12(c321f6dd/d) [3m].
Language: racket; memory limit: 1024 MB.
> (reconstruct
```



```
>
```

Determine language from source 7:2 639.69 MB

... and much, much more...