Trustworthy, Useful Languages for Probabilistic Modeling and Inference

---

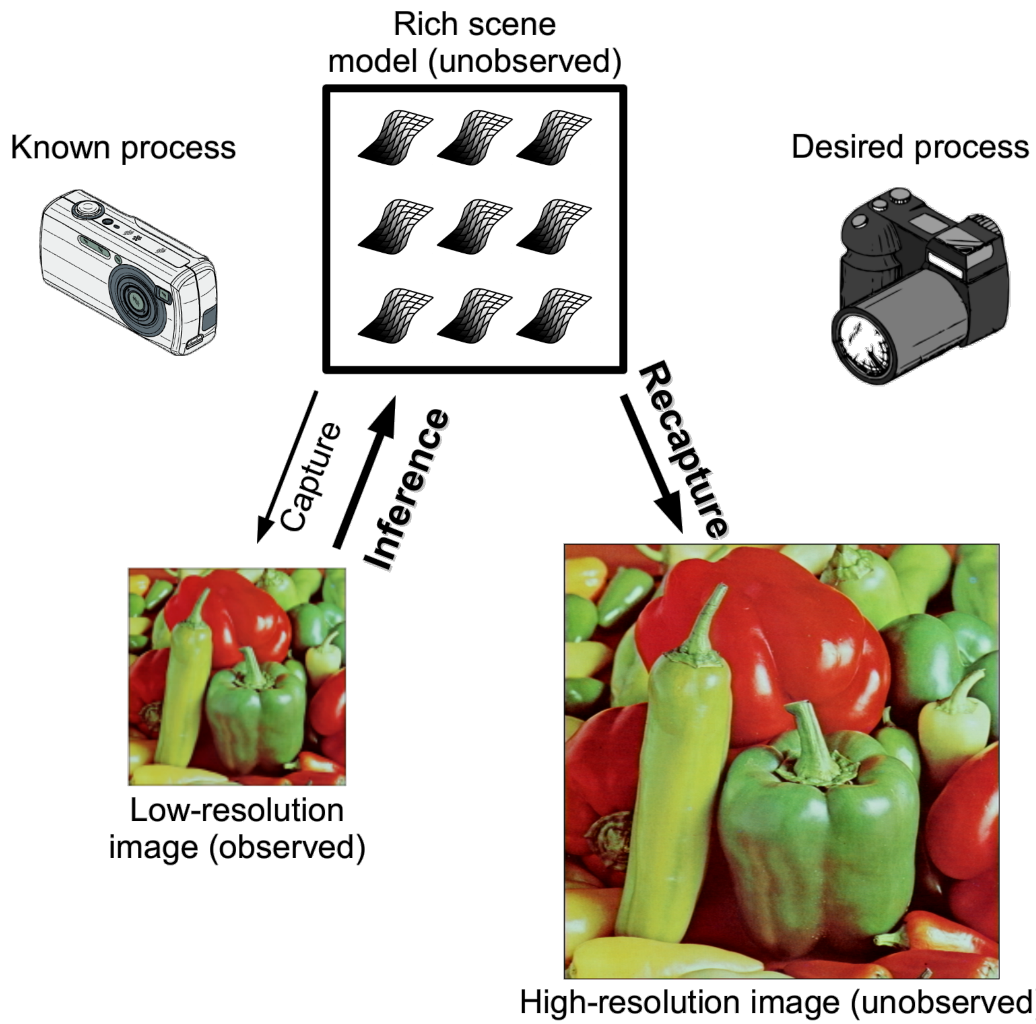
Neil Toronto

Dissertation Defense

Brigham Young University

2014/06/11

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

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

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

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

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

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

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

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

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

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

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

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

- 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

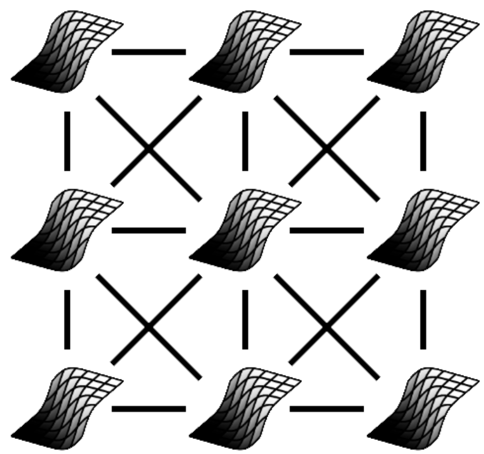
Only Mostly Satisfying

Problem 1: Still not sure the program is right

Only Mostly Satisfying

Problem 1: Still not sure the program is right

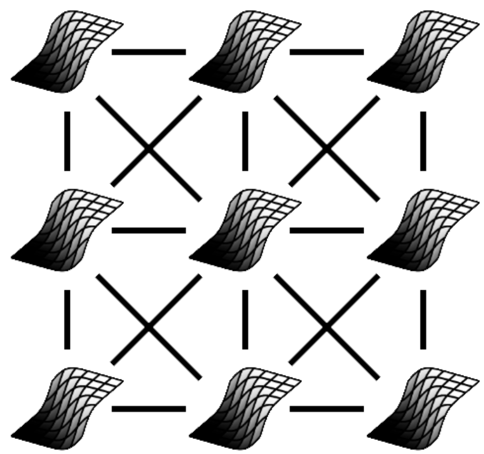
Problem 2: *smooth* edges instead of *discontinuous*



Only Mostly Satisfying

Problem 1: Still not sure the program is right

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

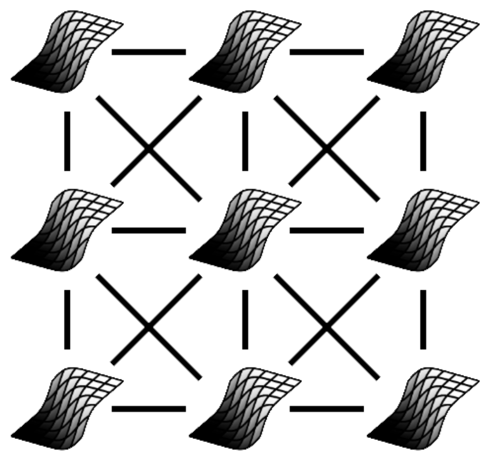


“To *approximate* blurring with a spatially varying point-spread function (PSF), we assign each facet a Gaussian PSF and convolve each analytically *before combining outputs*.”

Only Mostly Satisfying

Problem 1: Still not sure the program is right

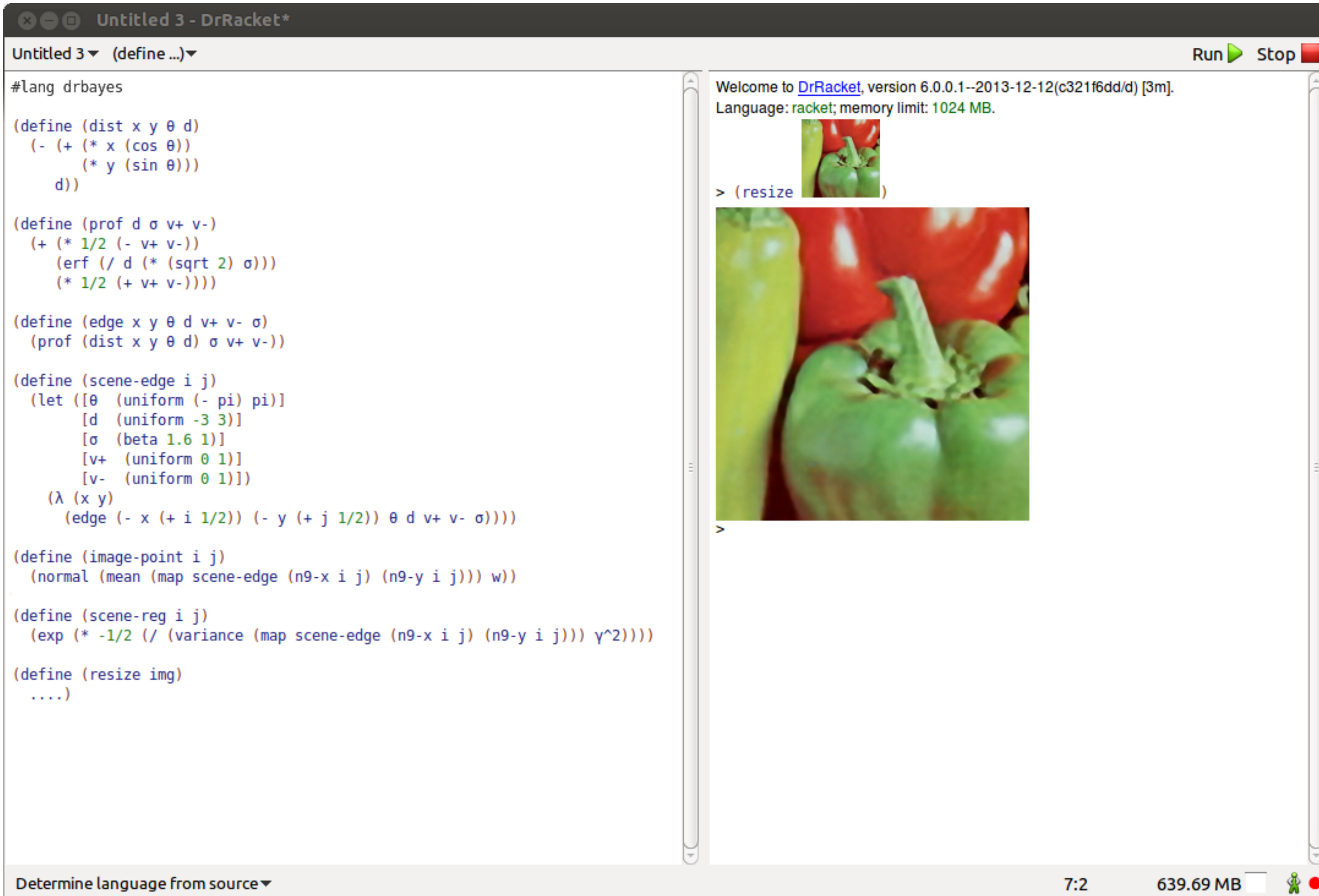
Problem 2: *smooth* edges instead of *discontinuous*



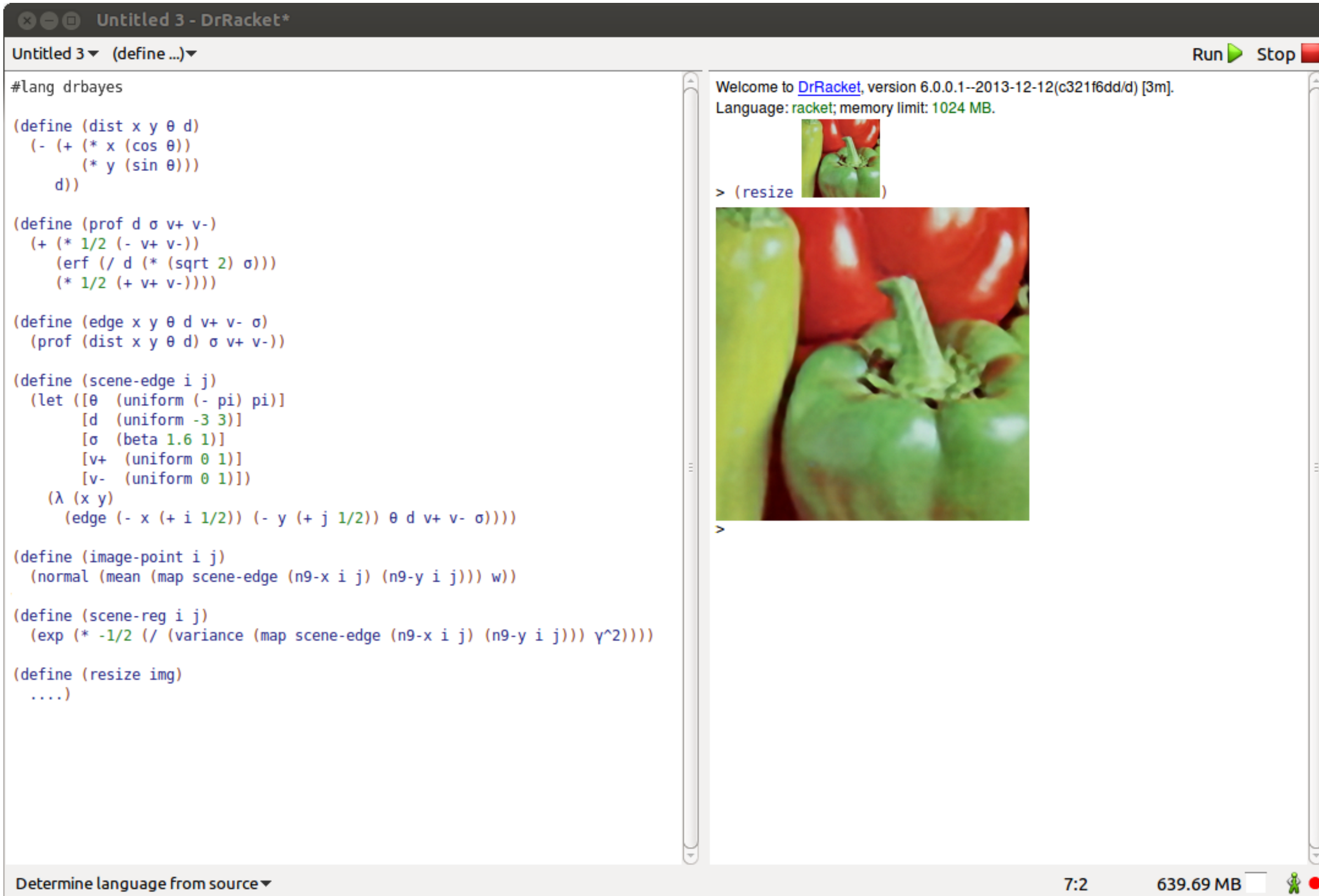
"To *approximate* blurring with a spatially varying point-spread function (PSF), we assign each facet a Gaussian PSF and convolve each analytically *before combining outputs*."

i.e. "We can't model it correctly so here's a hack."

Solution Idea: Probabilistic Language



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

Designed for Bayesian practice

Defined by a semantics (i.e. mathematically)

Prior Work

Defined by an implementation

Designed for Bayesian practice

Mimic human translation

Defined by a semantics (i.e. mathematically)

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

Mimic human translation

Can’t tell error from feature

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

Defined by a semantics (i.e. mathematically)

Prior Work

Defined by an implementation

Designed for Bayesian practice

Mimic human translation

Can't tell error from feature

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

Defined by a semantics (i.e. mathematically)

Designed for functional programmers or FP theorists

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	May not be implemented
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	May not be implemented
Can’t tell error from feature	Behavior is well-defined
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	May not be implemented
Can't tell error from feature	Behavior is well-defined
Limited: usually no recursion or loops; conditions $X = c$	Limited: usually finite distributions, no conditioning

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	May not be implemented
Can’t tell error from feature	Behavior is well-defined
Limited: usually no recursion or loops; conditions $X = c$	Limited: usually finite distributions, no conditioning

---

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

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

Simple Example Process

- Example process: Normal-Normal

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

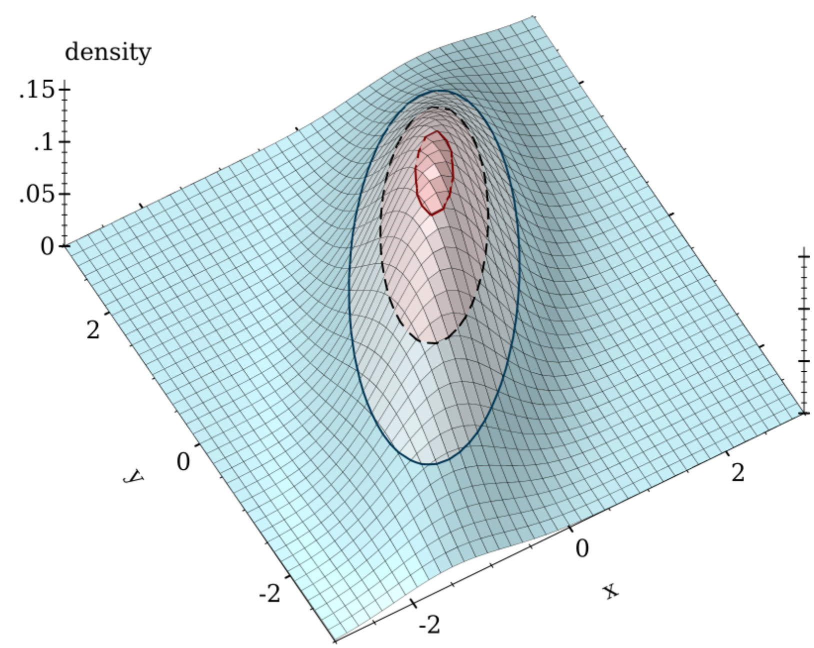
- Intuition: Sample X, then sample Y using X

Simple Example Process

- Example process: Normal-Normal

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

- Intuition: Sample X, then sample Y using X
- Density model $f : \mathbb{R} \times \mathbb{R} \to [0, \infty)$:



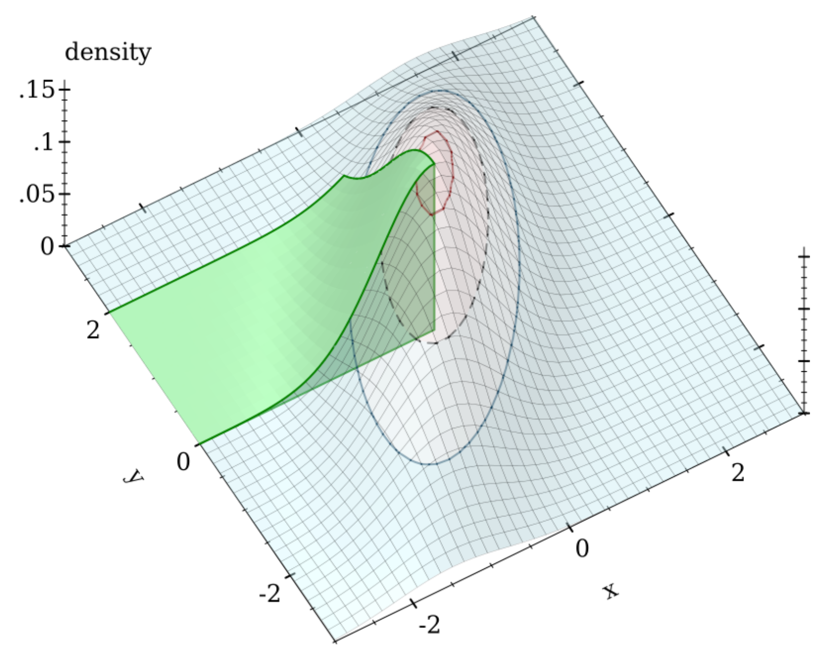
Simple Example Process

- Example process: Normal-Normal

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

- Intuition: Sample X, then sample Y using X

- Compute query $\Pr[X < 0, Y \in (0, 2)]$ by integrating:



Conditional Queries

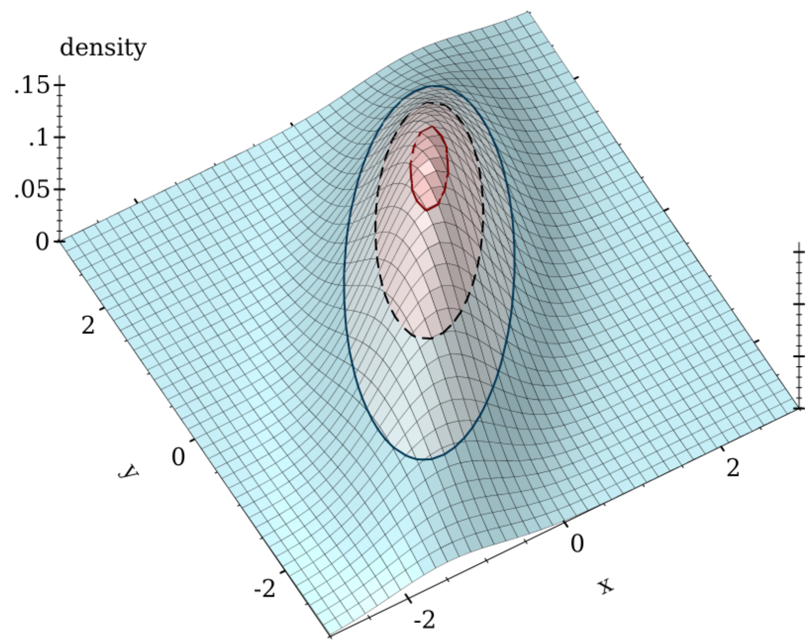
- Compute query $\Pr[X < 0 \mid Y = 1]$ using Bayes' law:

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

Conditional Queries

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

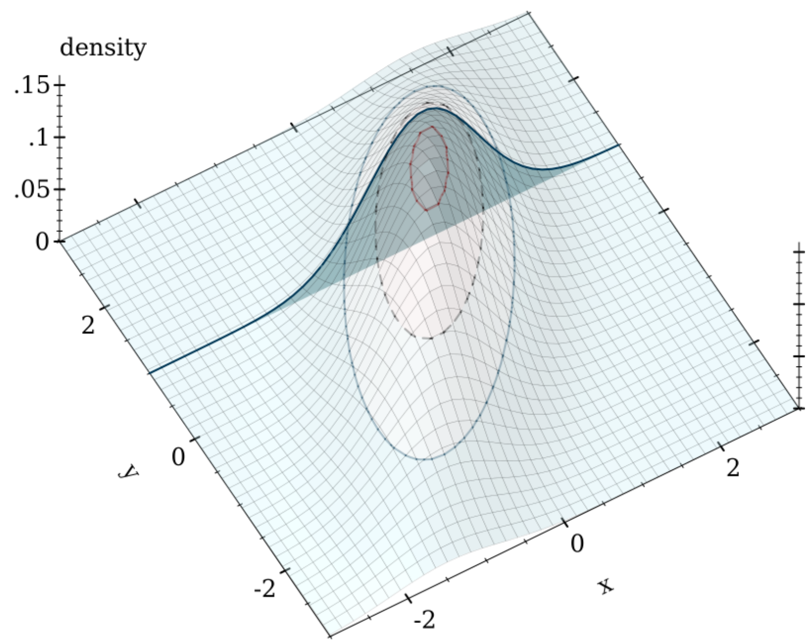
$$f(x \mid y) = \frac{f(x, y)}{\int_{-\infty}^{\infty} f(x, y)\ dx}$$



Conditional Queries

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

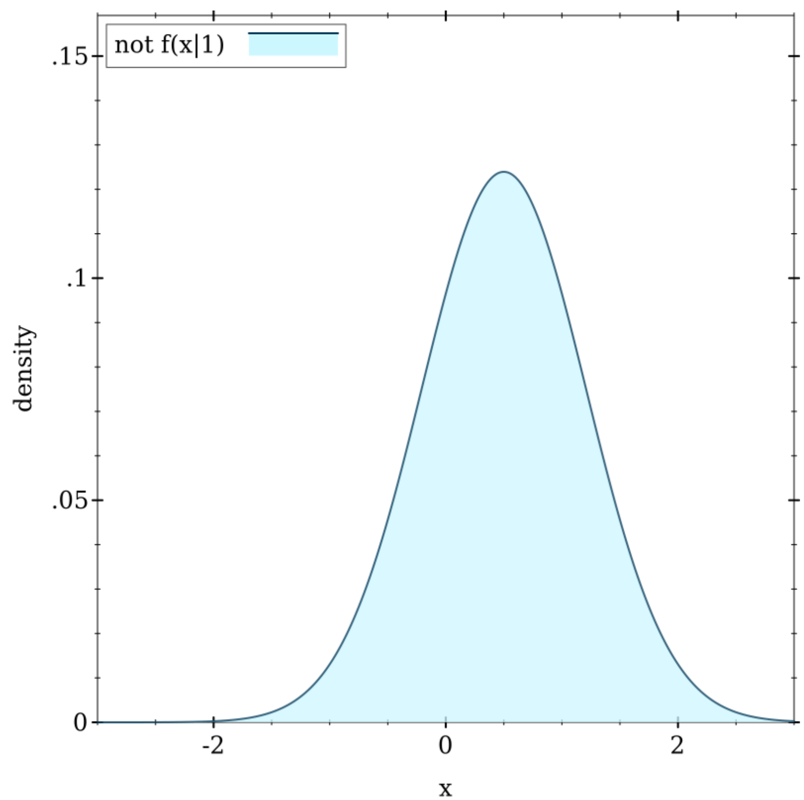
$$f(x \mid y) = \frac{f(x, y)}{\int_{-\infty}^{\infty} f(x, y)\ dx}$$



Conditional Queries

- Compute query $\Pr[X < 0 \,|\, Y = 1]$ using Bayes' law:

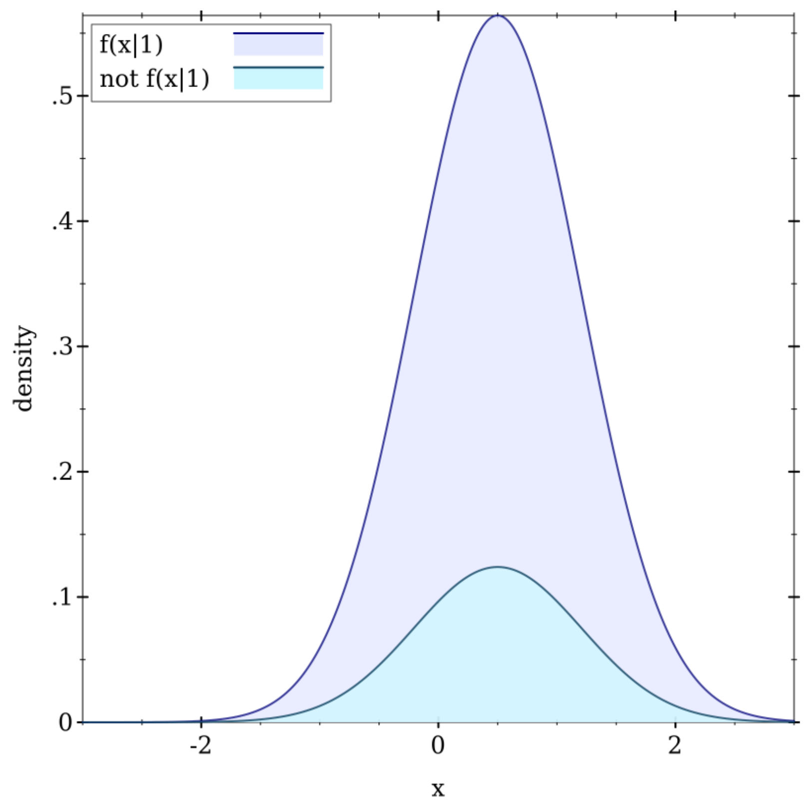
$$f(x \,|\, y) = \frac{f(x, y)}{\int_{-\infty}^{\infty} f(x, y)\ dx}$$



Conditional Queries

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

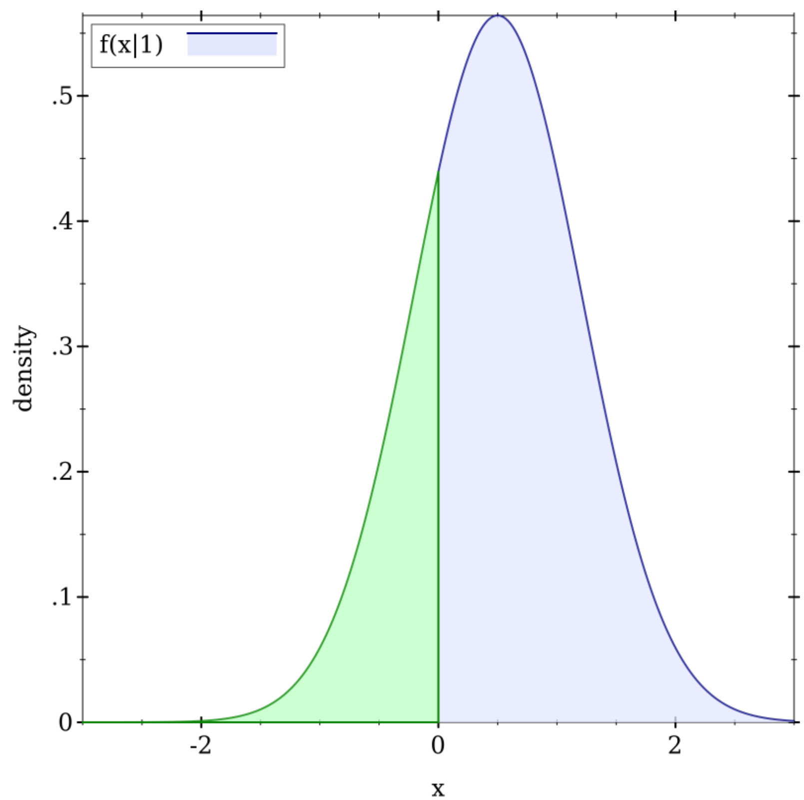
$$f(x \mid y) = \frac{f(x, y)}{\int_{-\infty}^{\infty} f(x, y)\ dx}$$



Conditional Queries

- Compute query $\Pr[X < 0 \,|\, Y = 1]$ using Bayes' law:

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



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)

$$\begin{aligned} Y &\sim \mathrm{Normal}(99, 1) \\ Y' &= \min(100, Y) \end{aligned}$$

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)

$$Y \sim \mathrm{Normal}(99, 1)$$
$$Y' = \min(100, Y)$$

  - 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)

$$Y \sim \mathrm{Normal}(99, 1)$$
$$Y' = \min(100, Y)$$

  - 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

$$X_n \sim \mathrm{Normal}(0, 1),\ n \in \mathbb{N}$$

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)

$$Y \sim \mathrm{Normal}(99, 1)$$
$$Y' = \min(100, Y)$$

  - 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

$$X_n \sim \mathrm{Normal}(0, 1), \; n \in \mathbb{N}$$

- 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:

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

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

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:

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

Measure-Theoretic Queries

- Specific query:

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

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

$$\Pr[e_1 \,|\, e_2] = \frac{\Pr[e_1, e_2]}{\Pr[e_2]}$$

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)

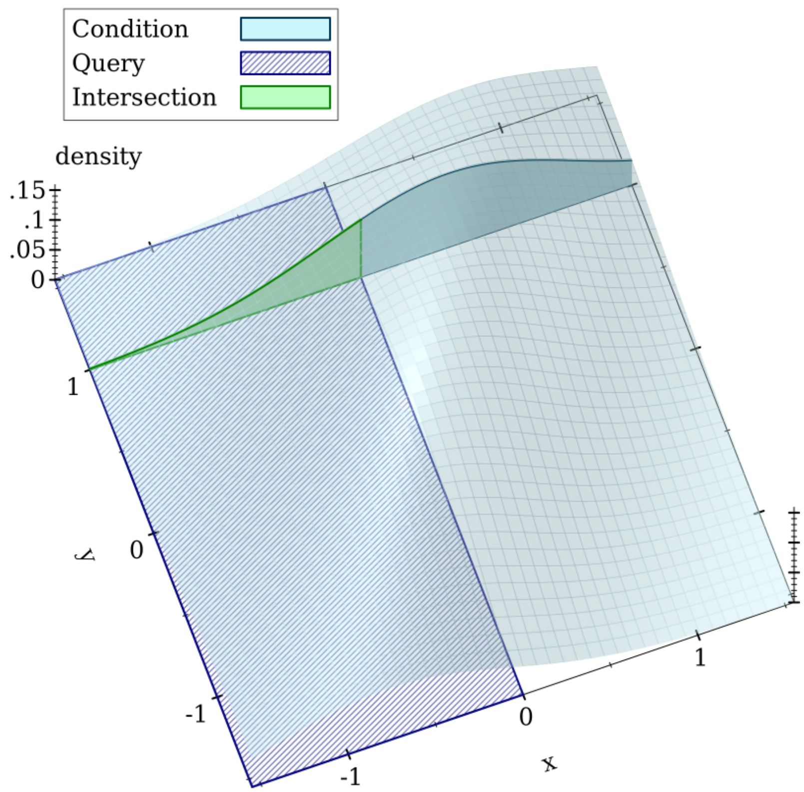
$$\Pr[X < 0 \,|\, Y = 1] \;=\; \lim_{\varepsilon \to 0} \Pr[X < 0 \,|\, |Y - 1| < \varepsilon]$$

Zero-Probability Conditions (Axial)

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

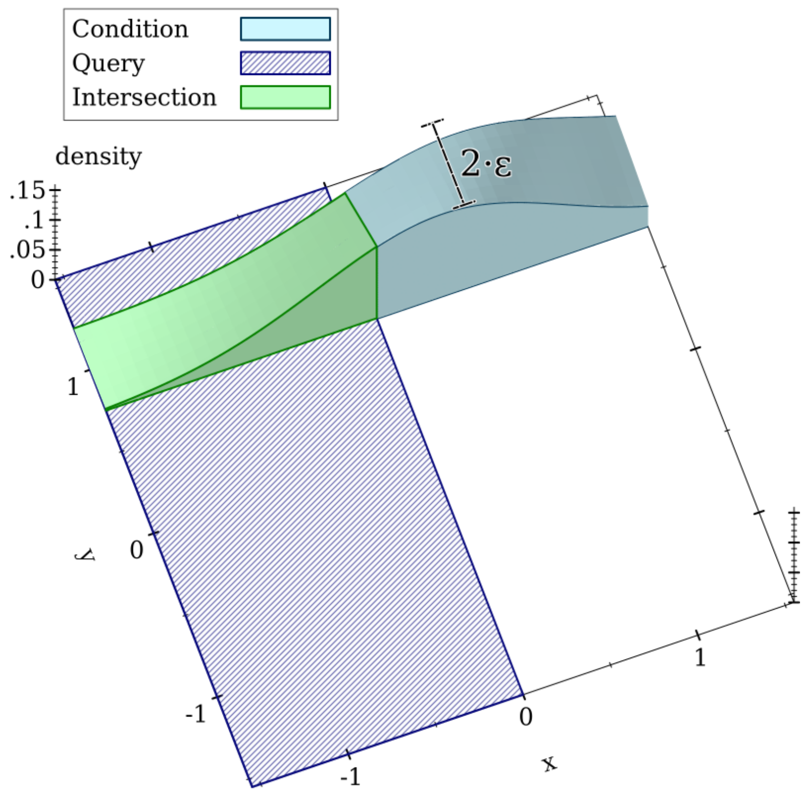
Zero-Probability Conditions (Axial)

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



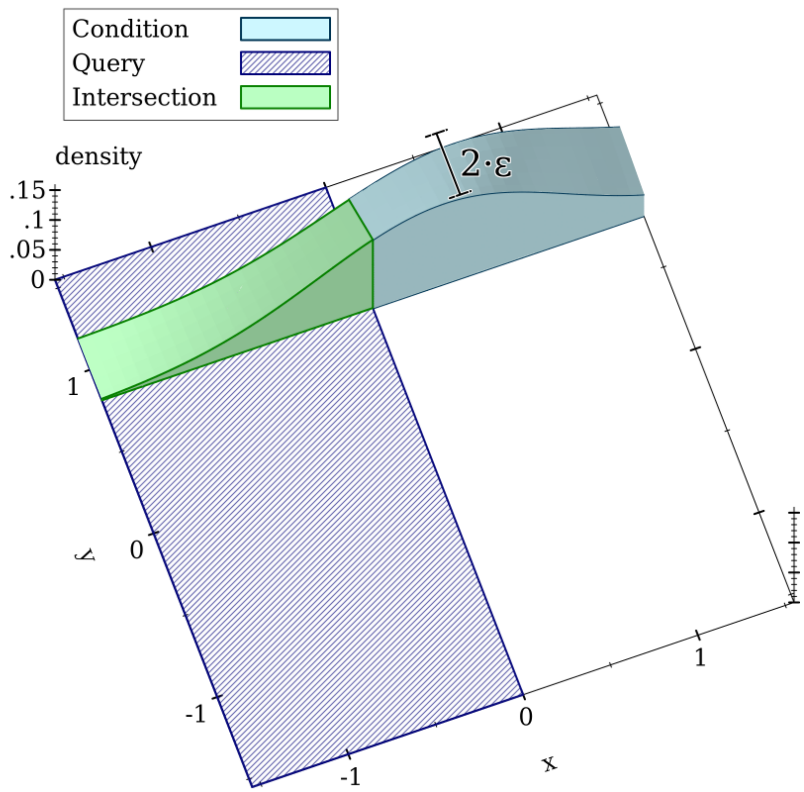
Zero-Probability Conditions (Axial)

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



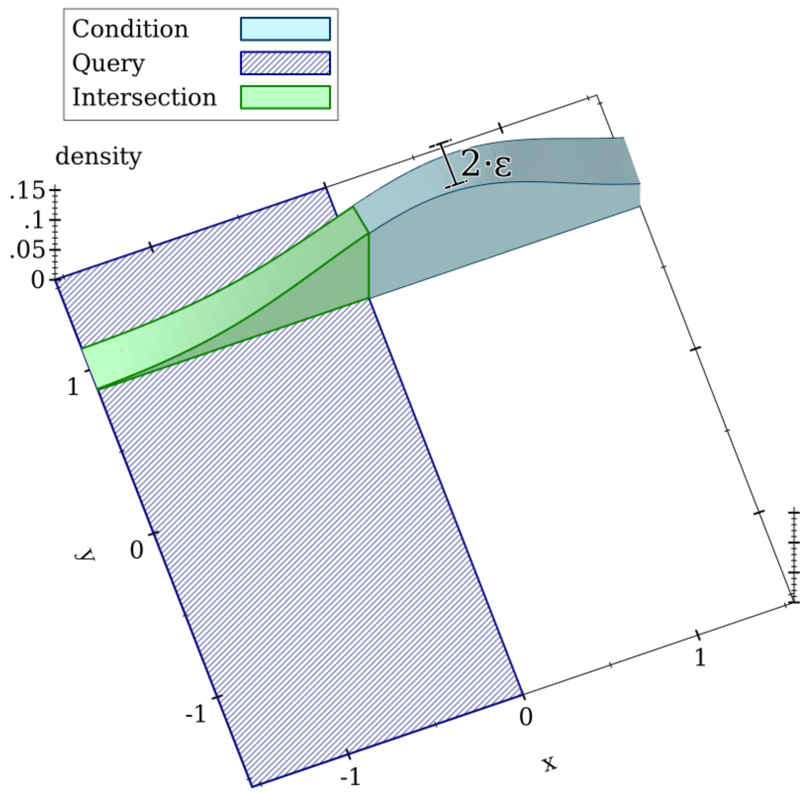
Zero-Probability Conditions (Axial)

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



Zero-Probability Conditions (Axial)

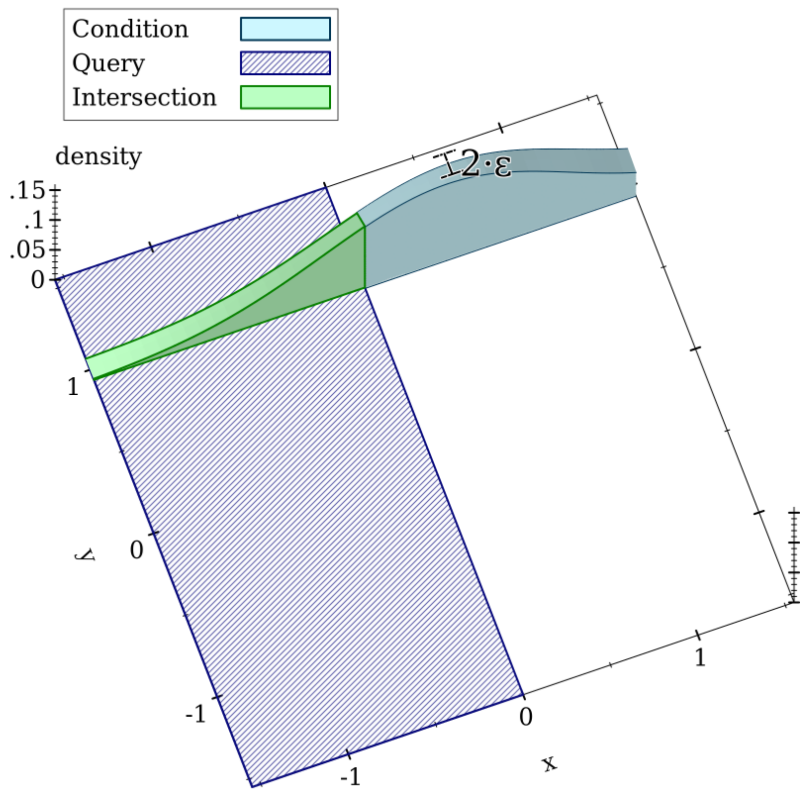
$$\Pr[X < 0 \mid Y = 1] = \lim_{\varepsilon \to 0} \Pr[X < 0 \mid |Y - 1| < \varepsilon]$$
$$= \lim_{\varepsilon \to 0} \frac{\Pr[X < 0, |Y - 1| < \varepsilon]}{\Pr[|Y - 1| < \varepsilon]}$$



Zero-Probability Conditions (Axial)

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

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



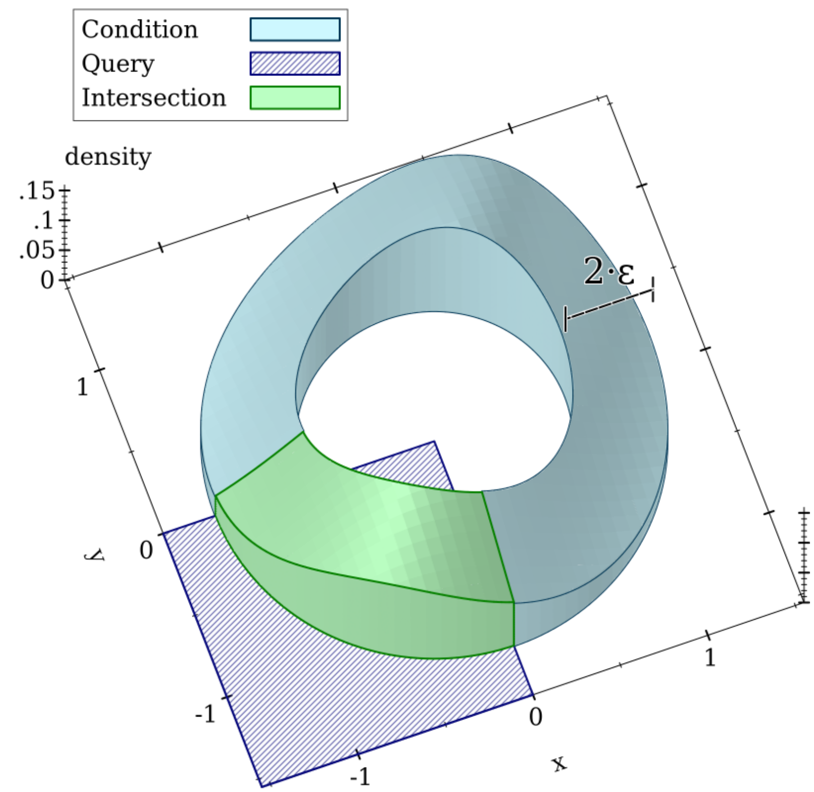
Zero-Probability Conditions (Circular)

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

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

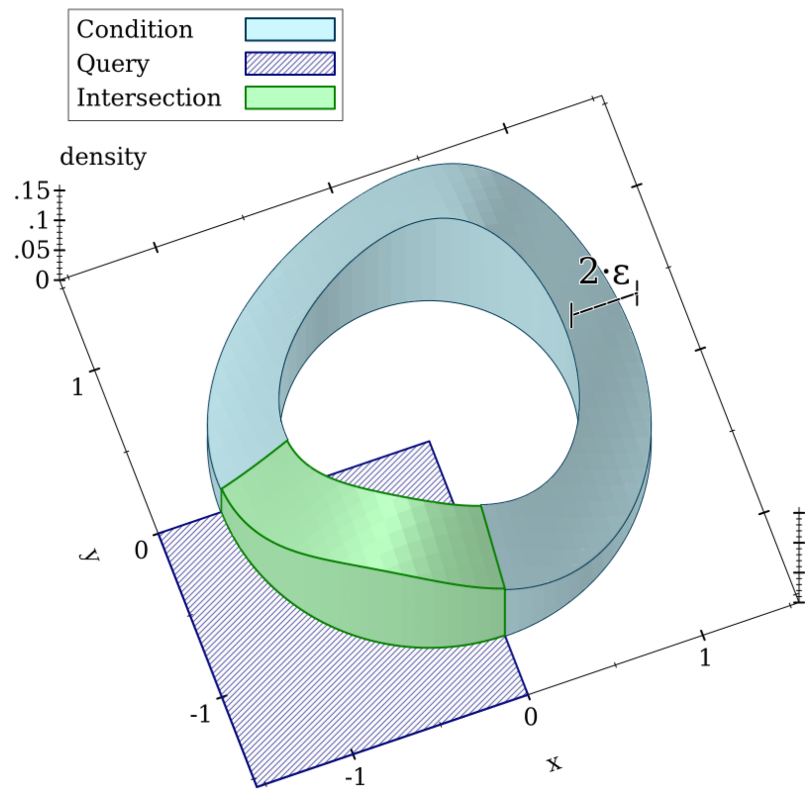
Zero-Probability Conditions (Circular)

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



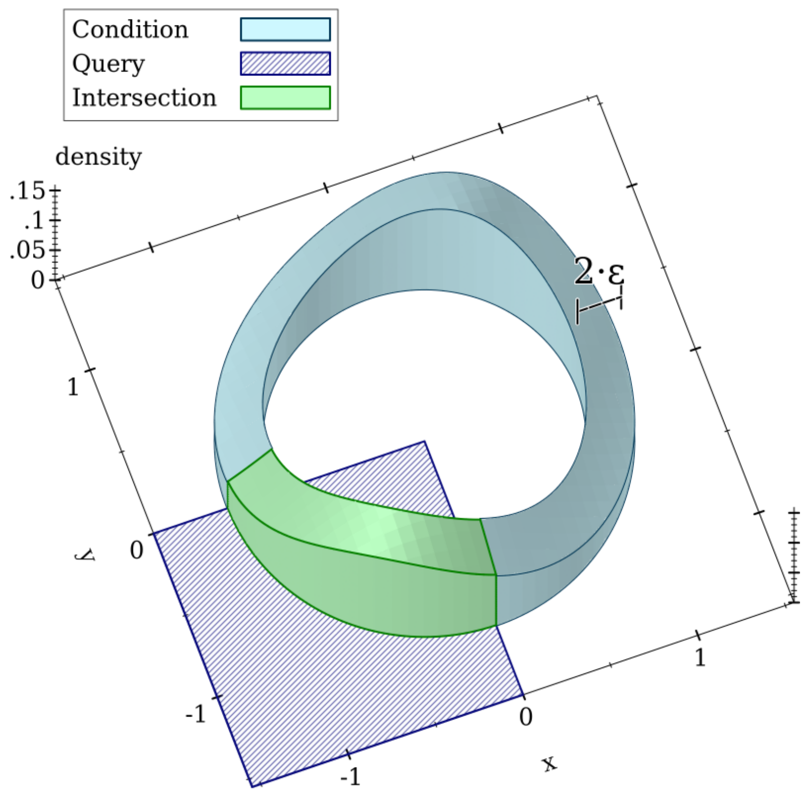
Zero-Probability Conditions (Circular)

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



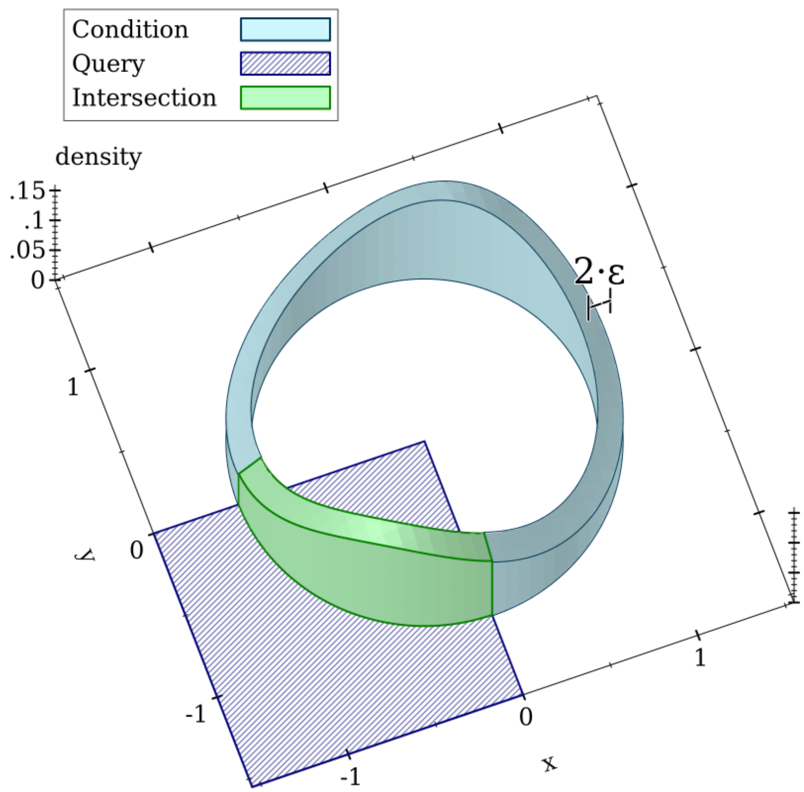
Zero-Probability Conditions (Circular)

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



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 $\mathrm{Pr}[\cdot]$ are an abstraction boundary hiding Ω 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 Ω and P, so we can choose convenient ones

---

A **uniform random source** model:

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

Contribution: Don't Integrate, Compute Backwards (1)

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

---

A **uniform random source** model:

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

where $F : \mathbb{R} \to [0,1]$ 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 Ω and P, so we can choose convenient ones

---

A **uniform random source** model:

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

where $F : \mathbb{R} \to [0,1]$ 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:

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

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

Contribution: Don't Integrate, Compute Backwards (2)

- 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 probabilities by computing **preimage** areas

Contribution: Don't Integrate, Compute Backwards (2)

- 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 probabilities by computing **preimage** areas
  - Compute conditional probabilities as quotients of **preimage** areas

Contribution: Don't Integrate, Compute Backwards (2)

- 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 probabilities by computing **preimage** areas
  - Compute conditional probabilities as quotients of **preimage** areas
- Is this really more feasible than integrating?

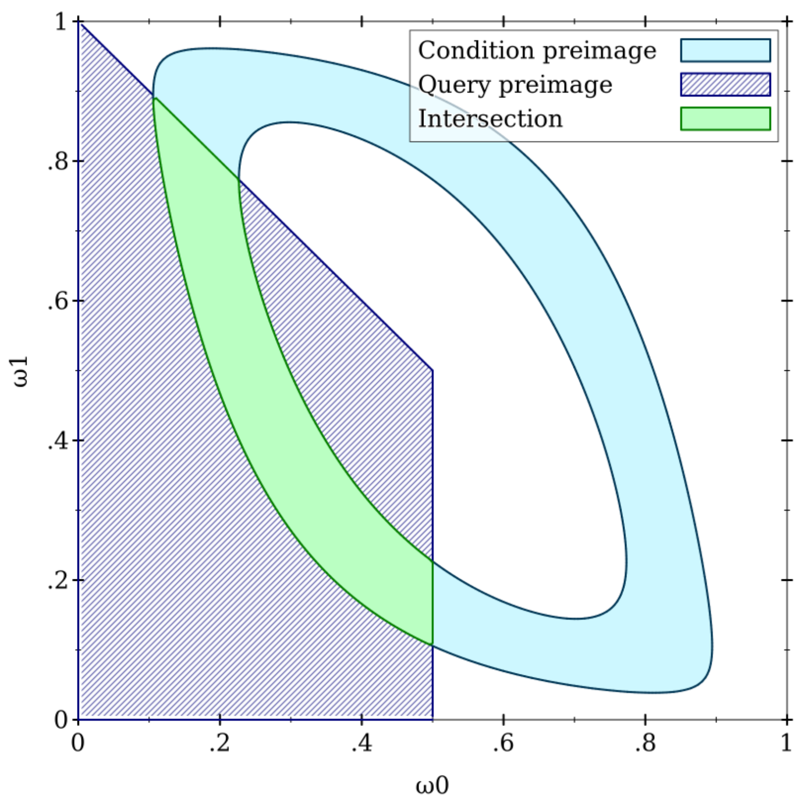
Queries Using Preimages

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

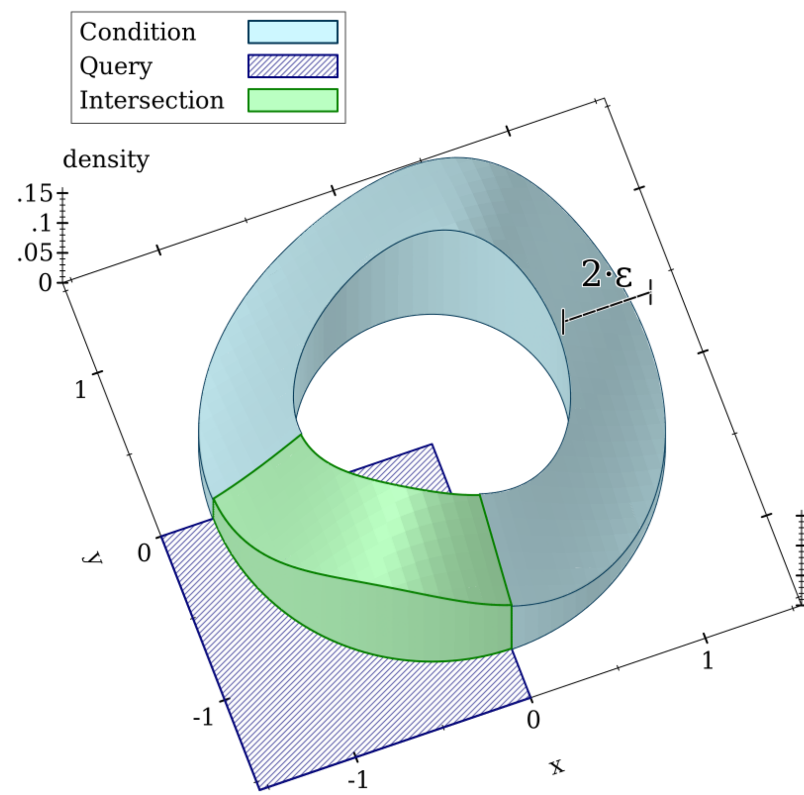
Queries Using Preimages

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

Uniform Random Source



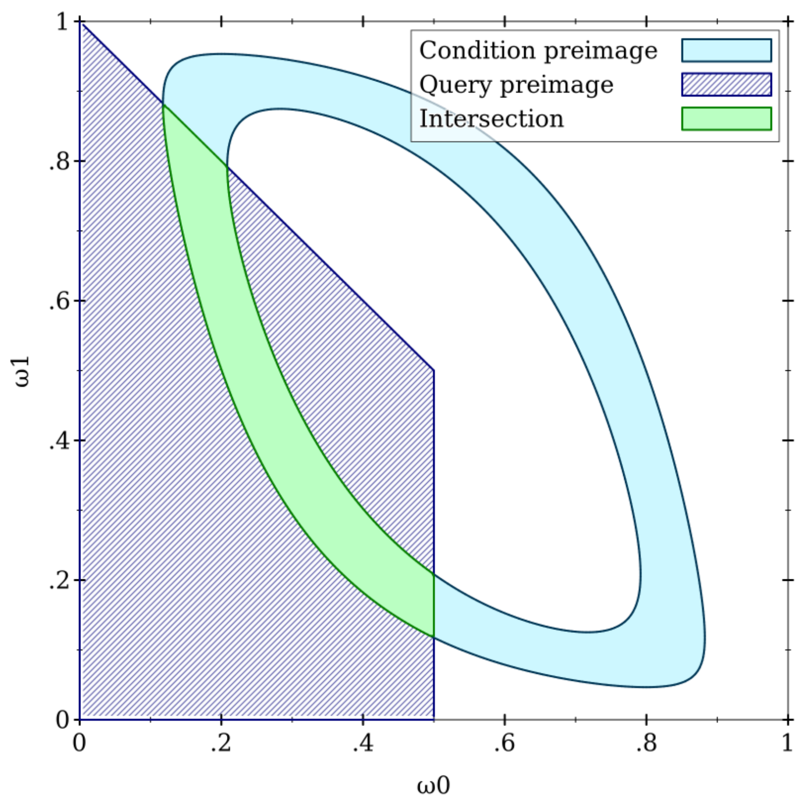
Original Model



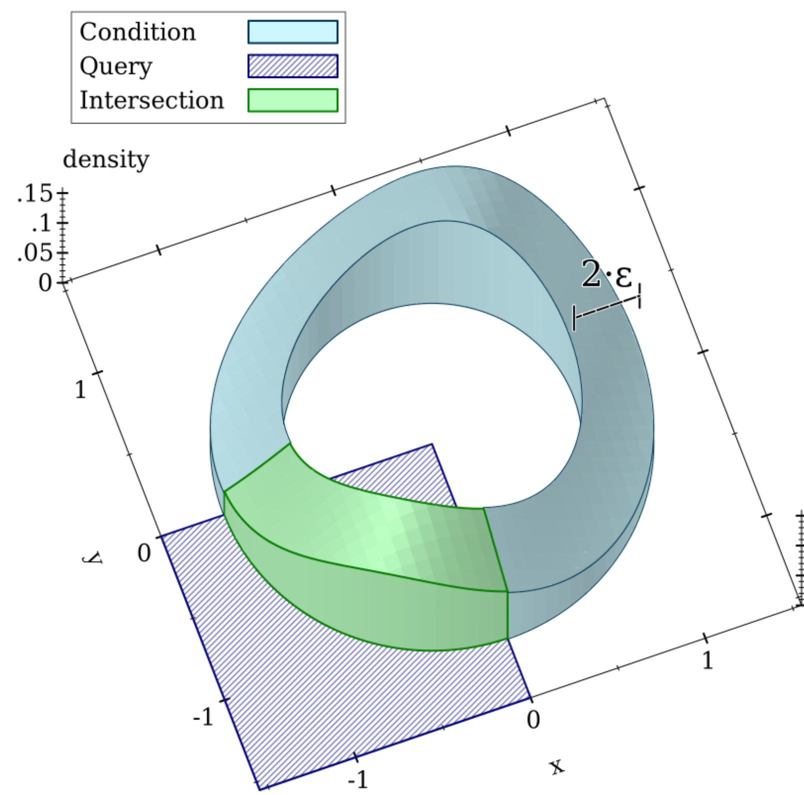
Queries Using Preimages

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

Uniform Random Source



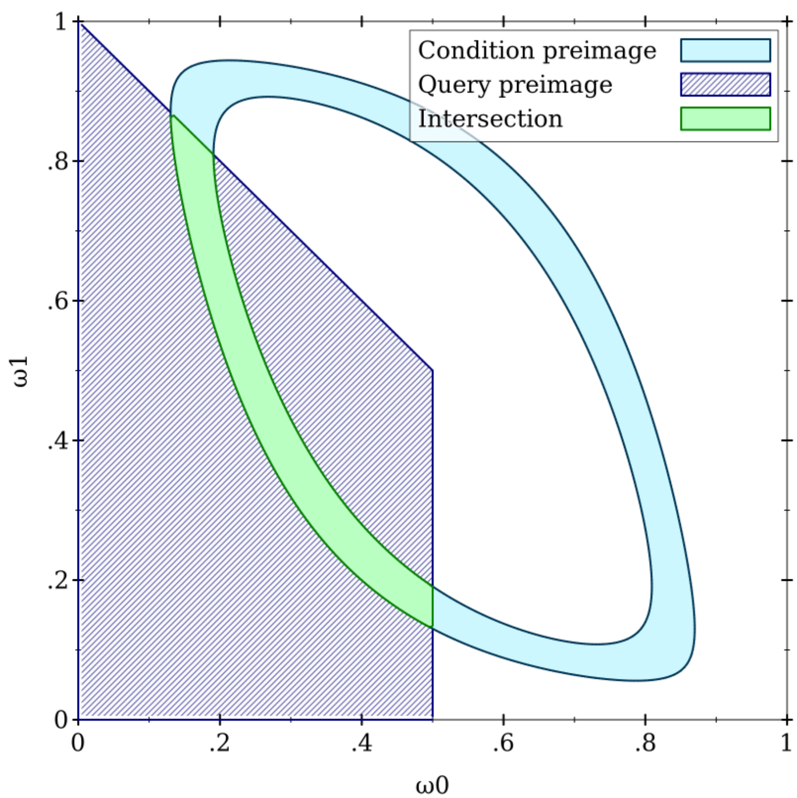
Original Model



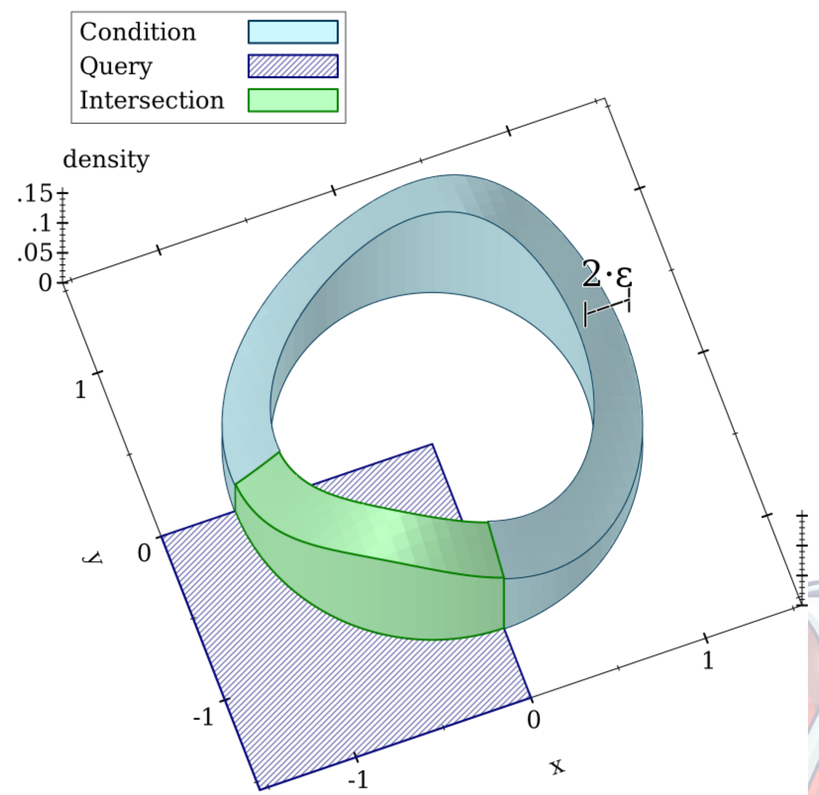
Queries Using Preimages

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

Uniform Random Source



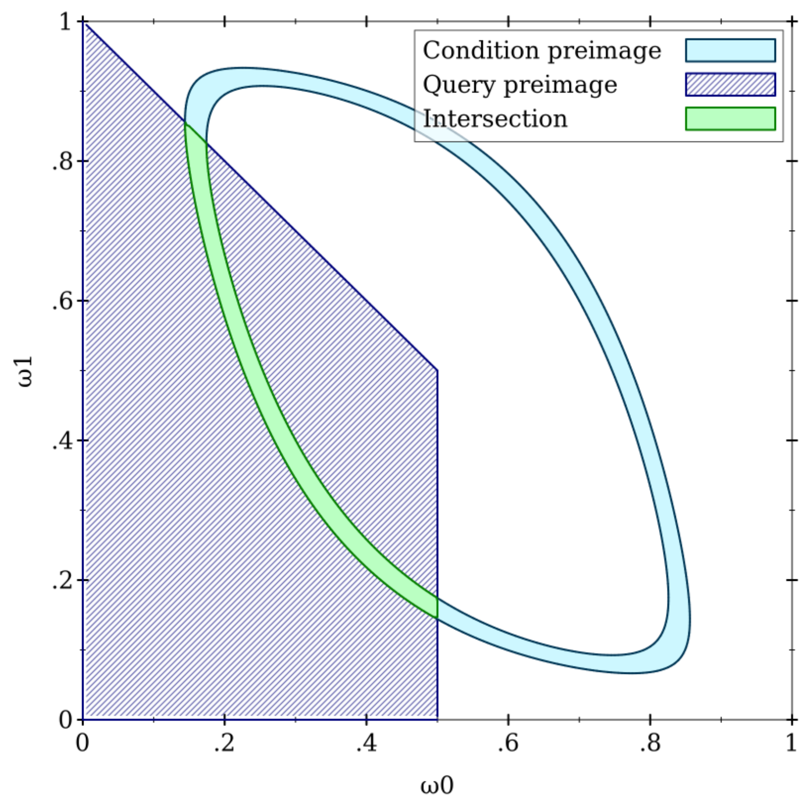
Original Model



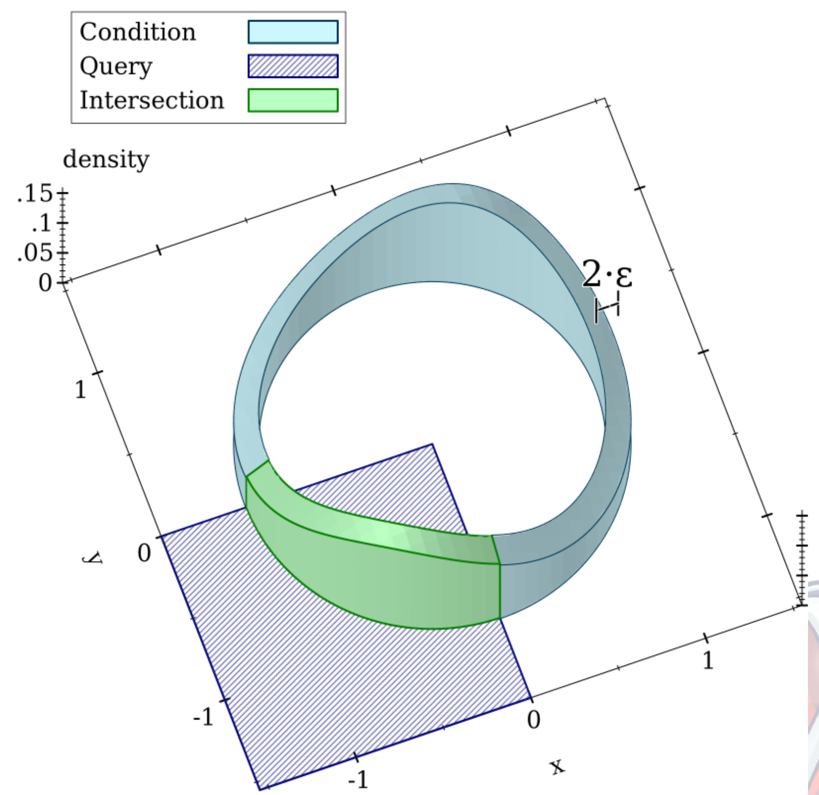
Queries Using Preimages

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

Uniform Random Source



Original Model



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...

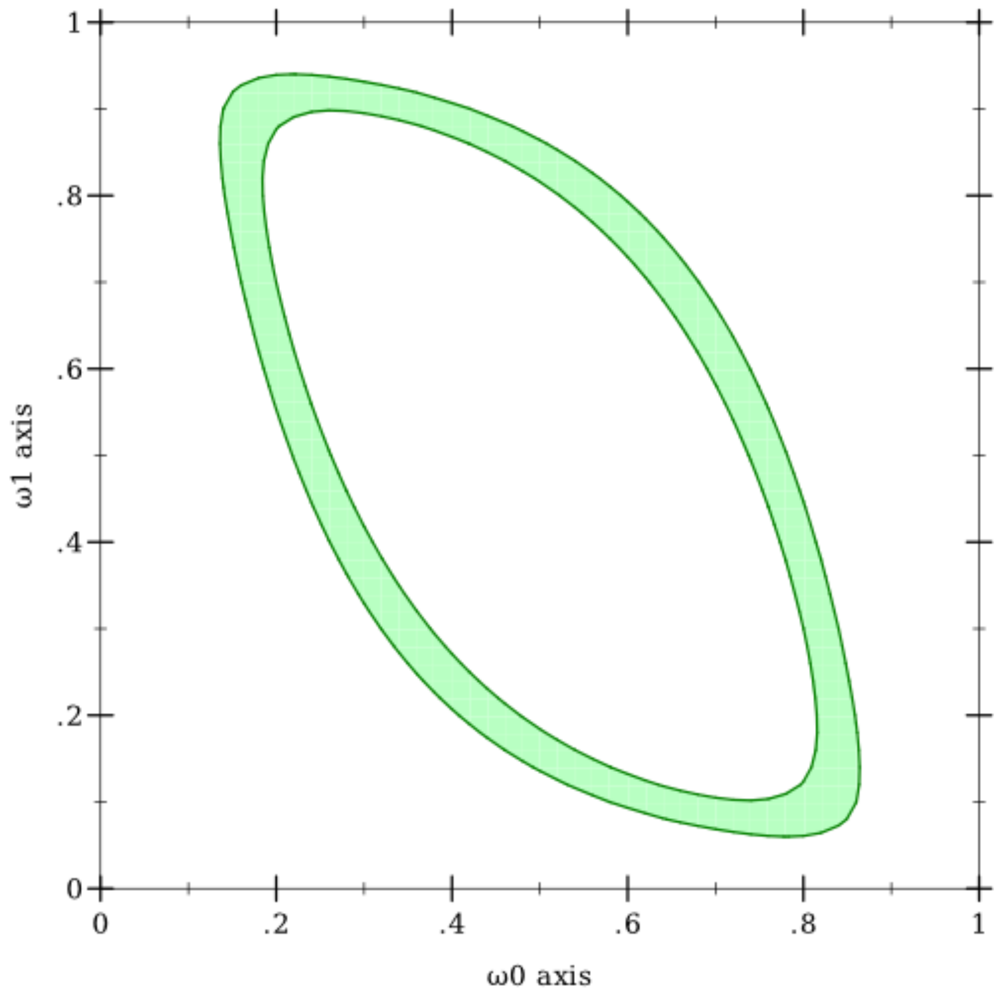
- Seems like we need:
  - Standard interpretation of programs as pure functions from a random source
  - Efficient way to compute preimage sets
  - Efficient representation of arbitrary sets
  - Efficient way to compute areas of preimage sets
  - Proof of correctness w.r.t. standard interpretation

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...

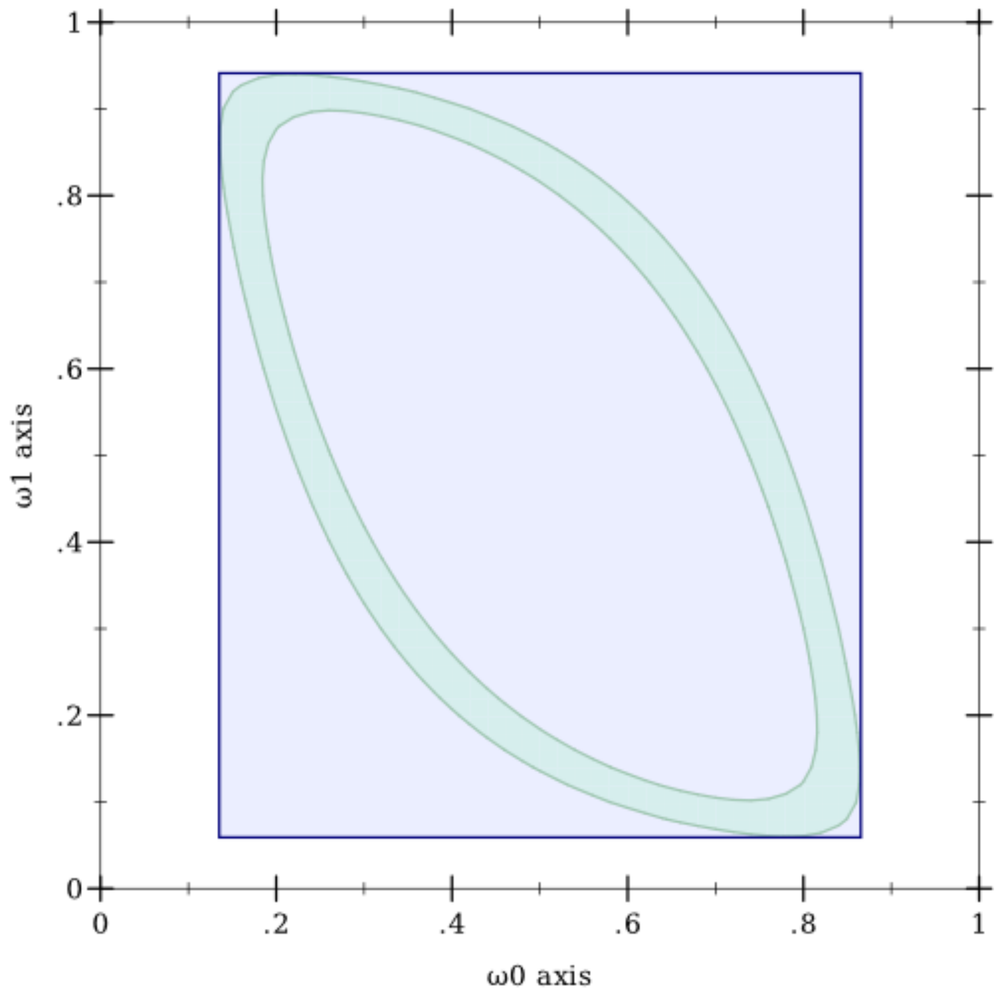
What About Approximating?

Conservative approximation with rectangles:



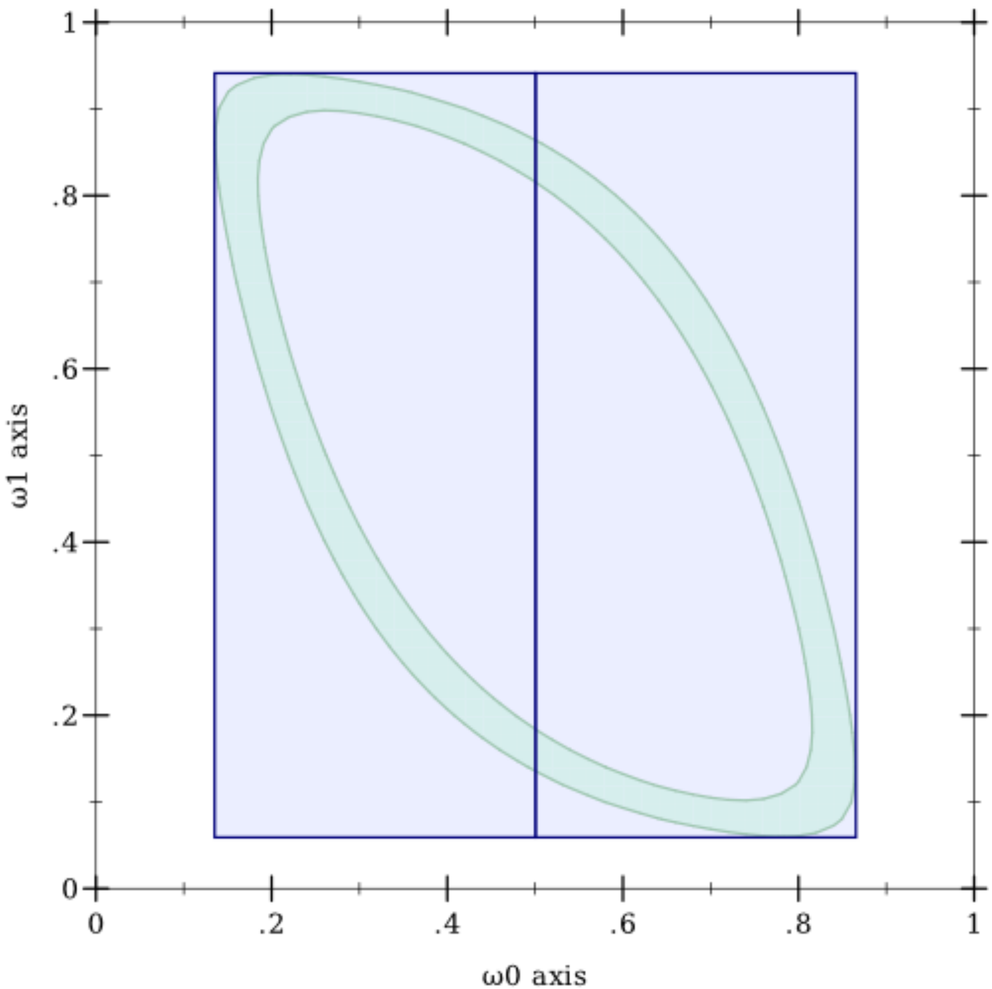
What About Approximating?

Conservative approximation with rectangles:



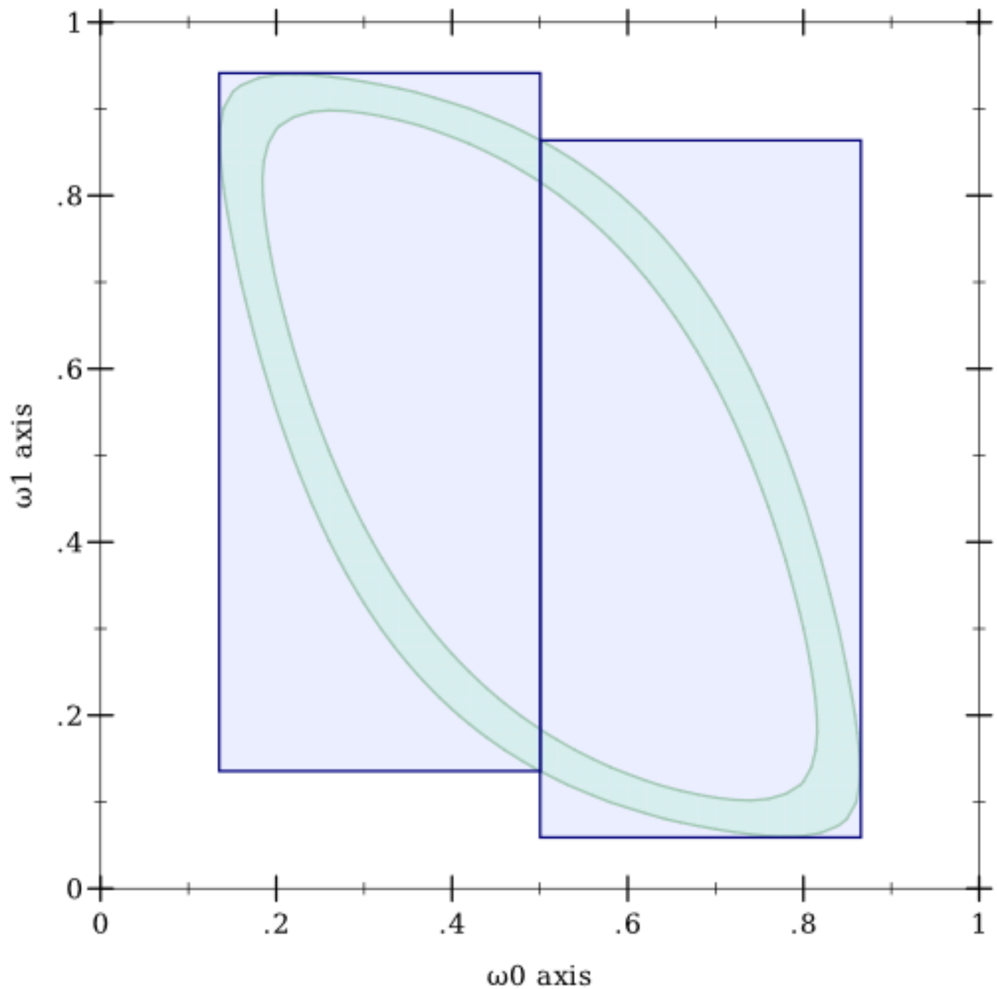
What About Approximating?

Restricting preimages to rectangular subdomains:



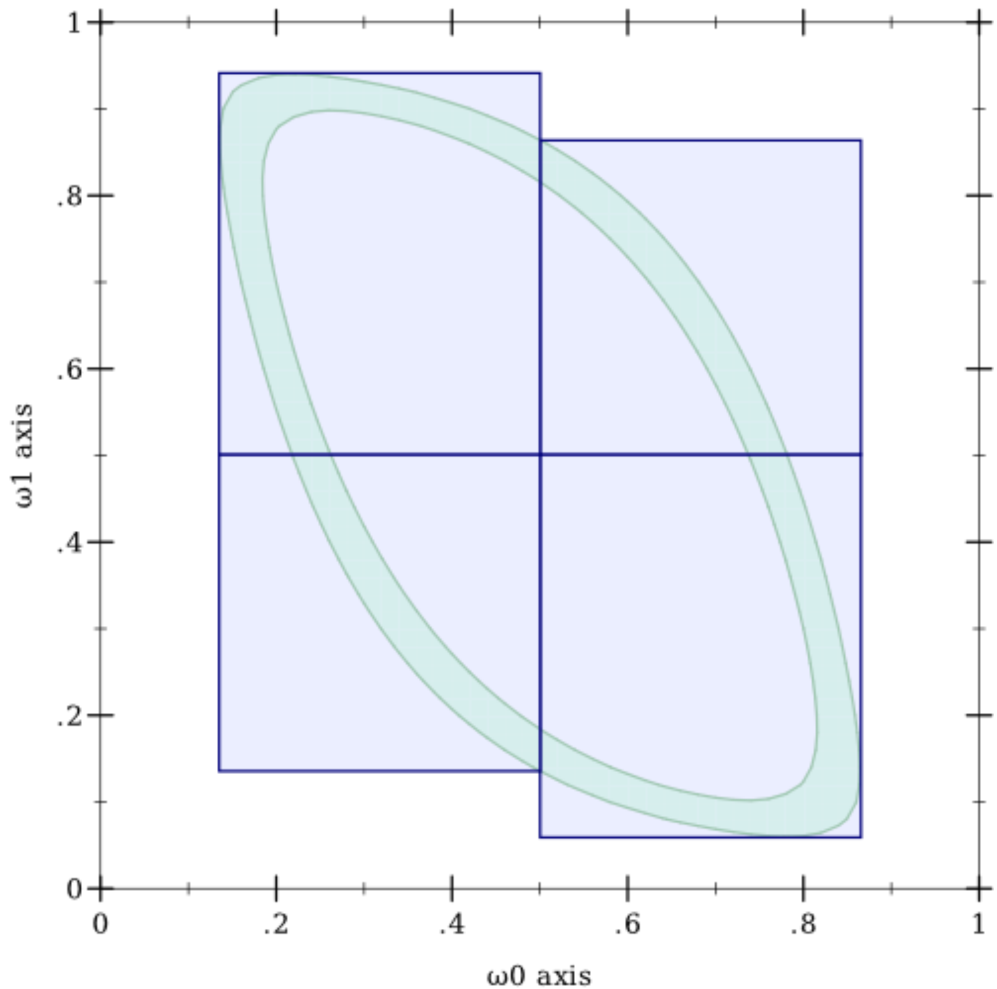
What About Approximating?

Restricting preimages to rectangular subdomains:



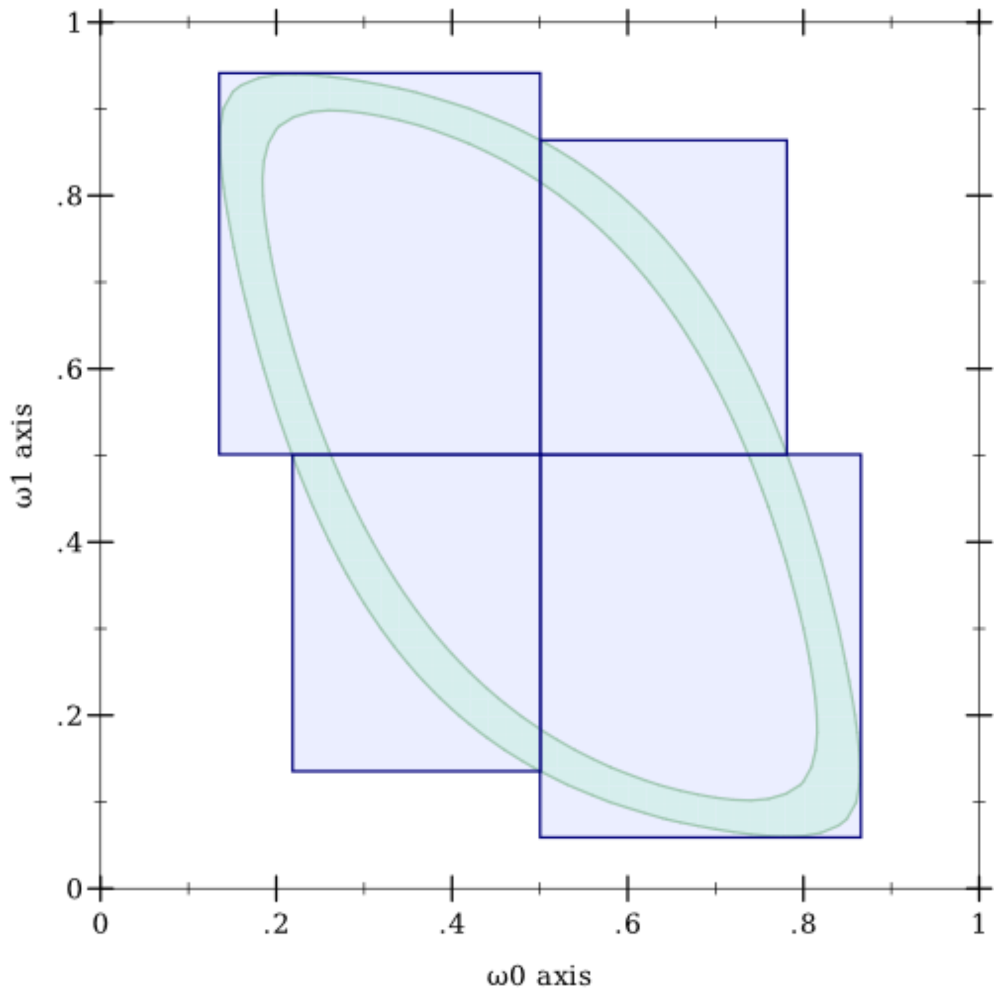
What About Approximating?

Restricting preimages to rectangular subdomains:



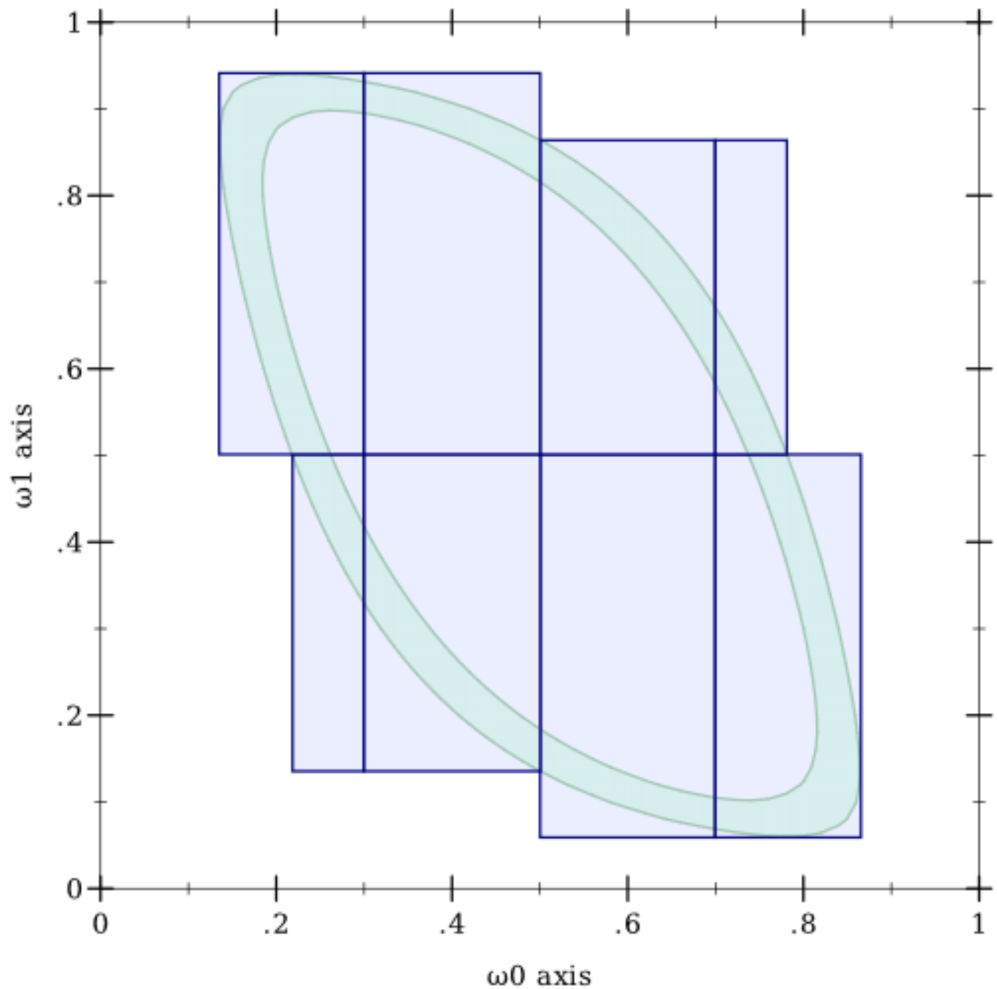
What About Approximating?

Restricting preimages to rectangular subdomains:



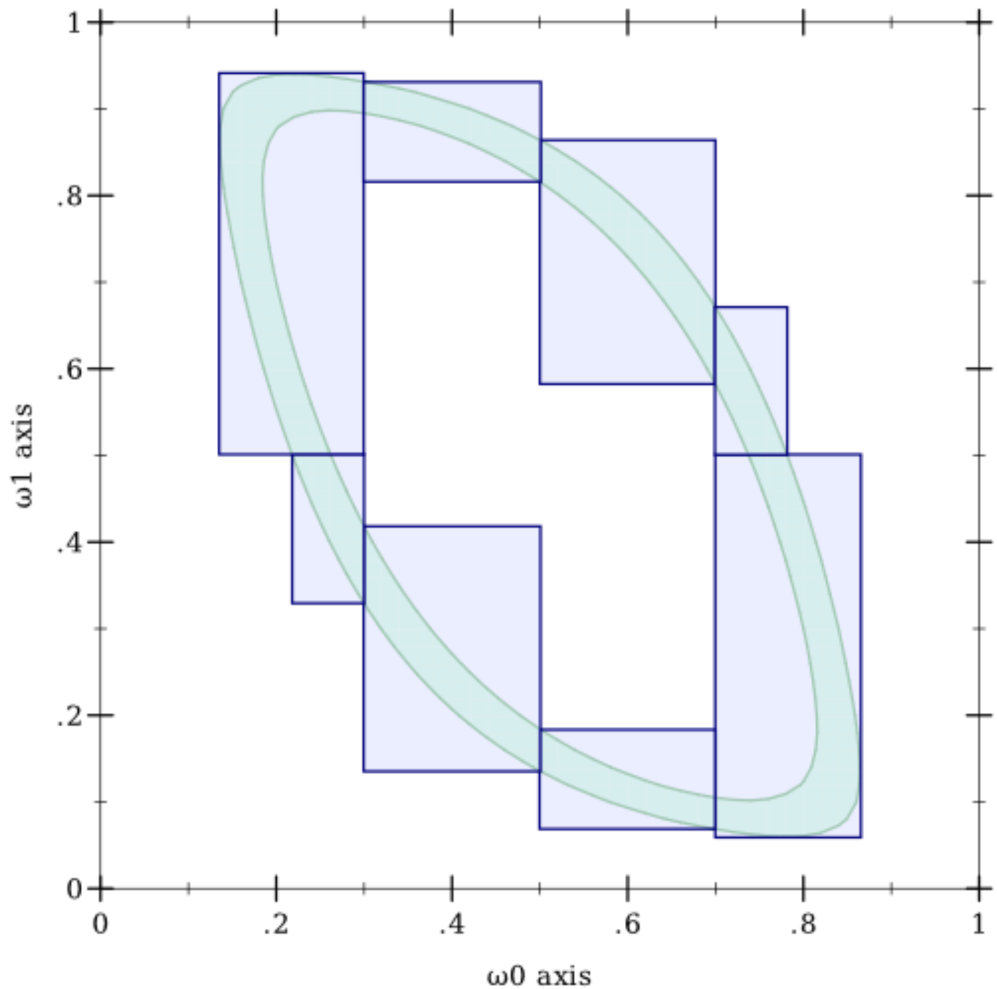
What About Approximating?

Restricting preimages to rectangular subdomains:



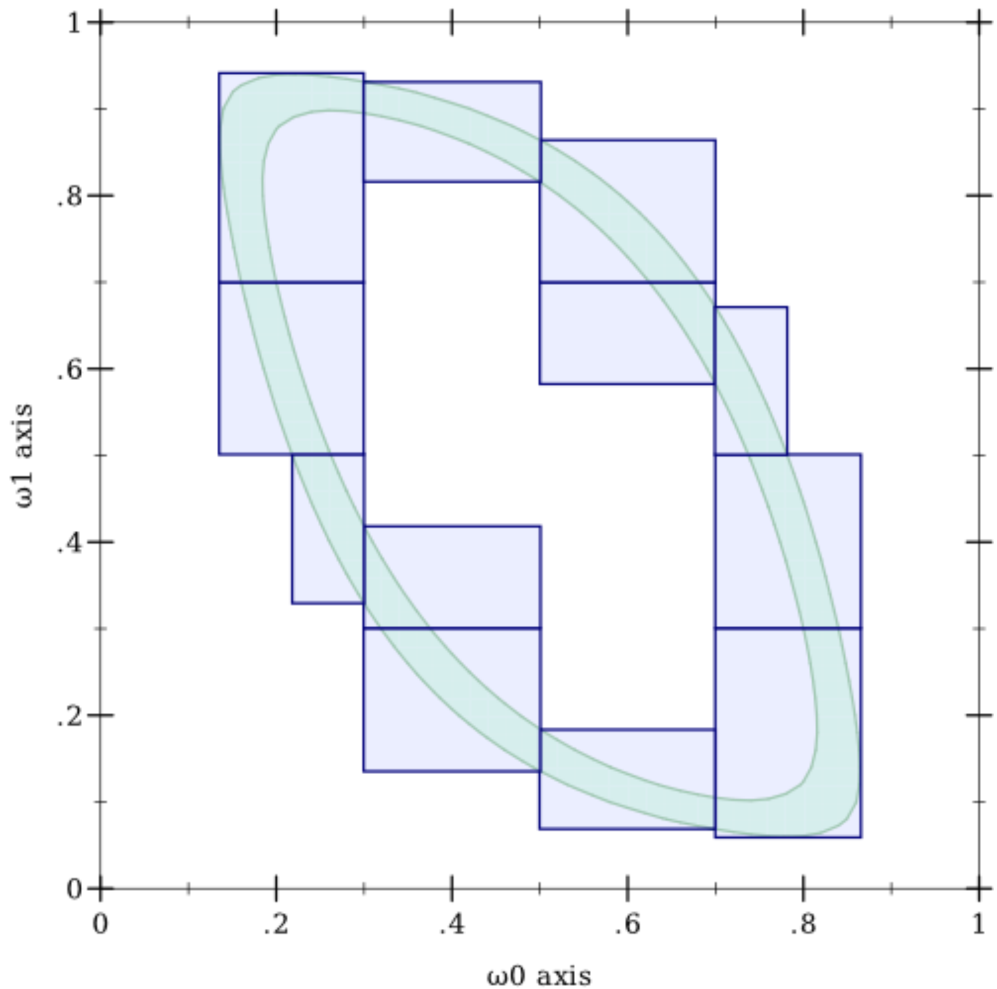
What About Approximating?

Restricting preimages to rectangular subdomains:



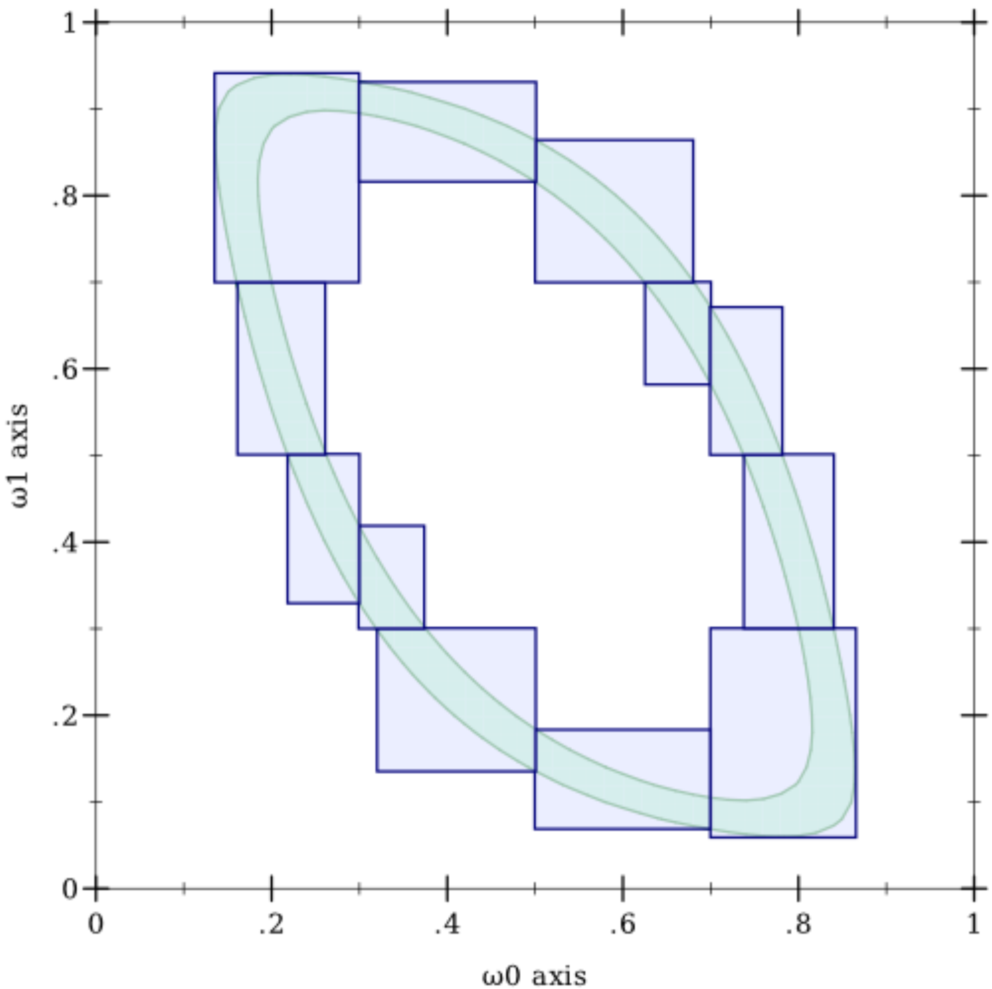
What About Approximating?

Restricting preimages to rectangular subdomains:



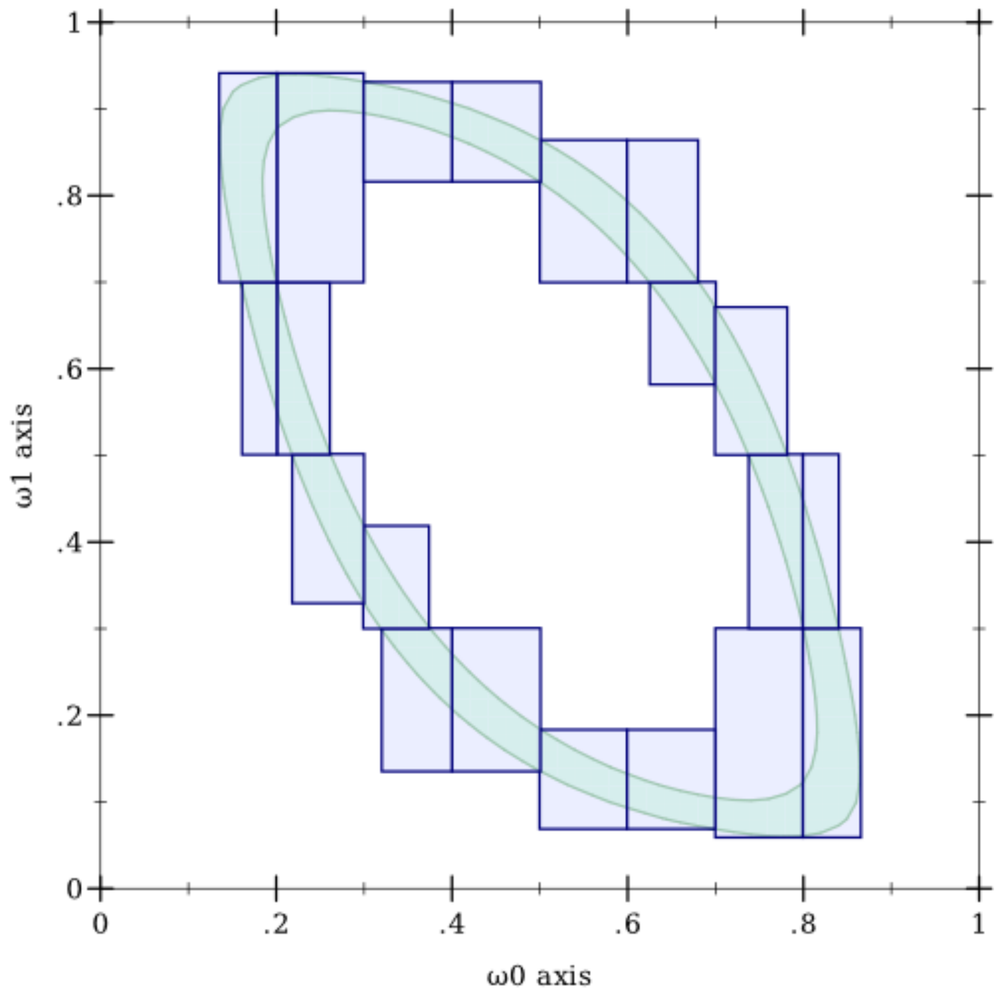
What About Approximating?

Restricting preimages to rectangular subdomains:



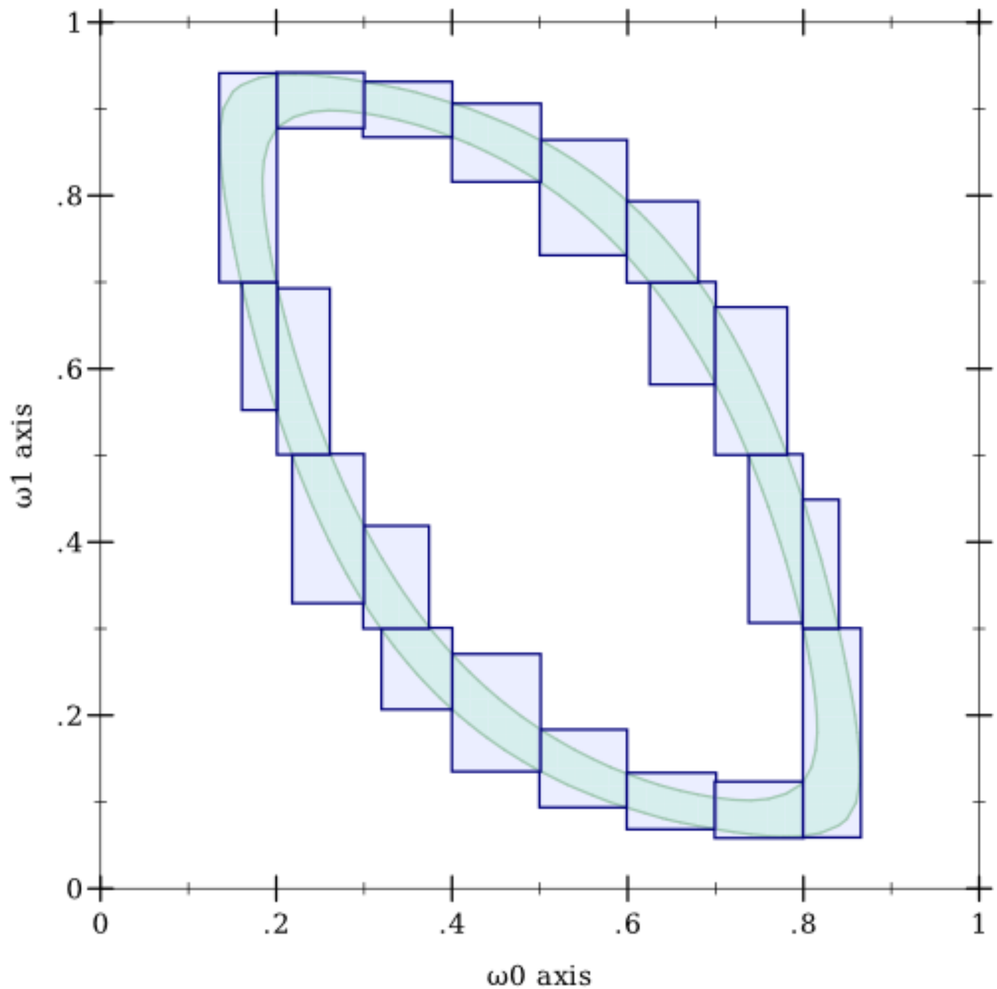
What About Approximating?

Restricting preimages to rectangular subdomains:



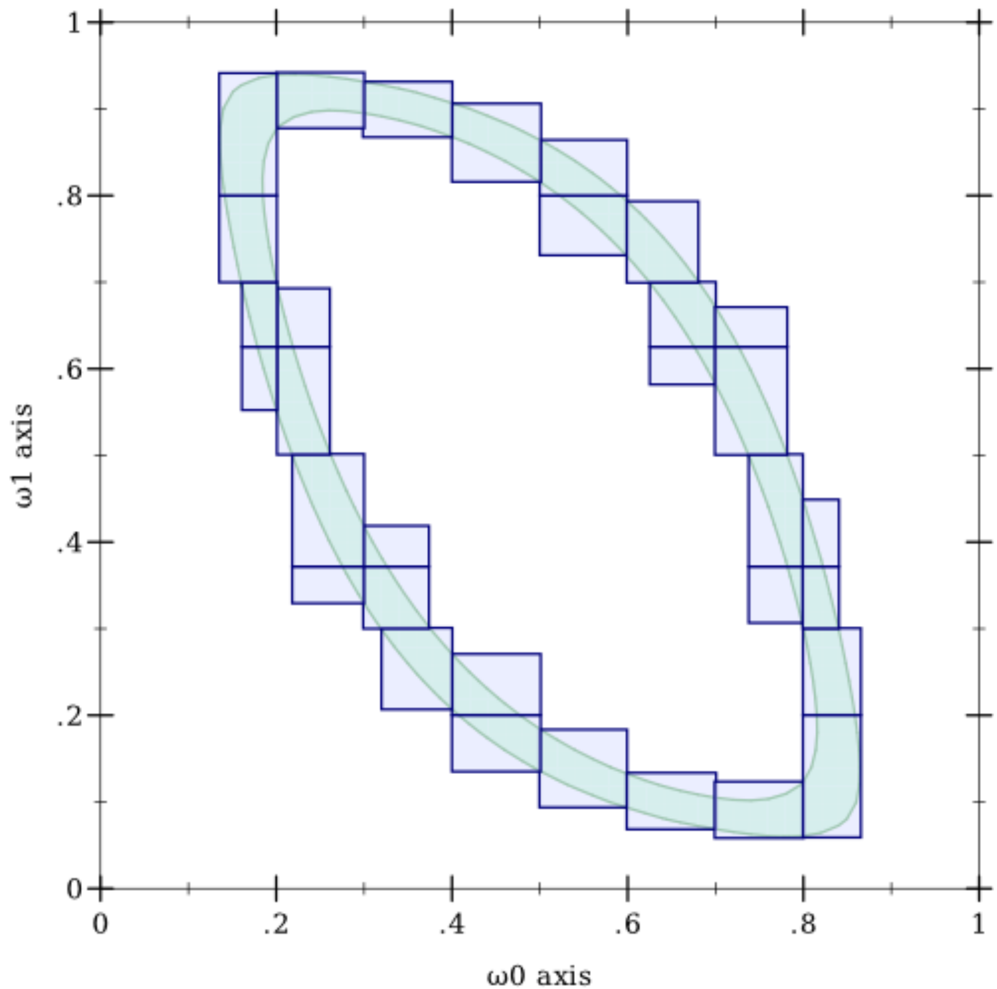
What About Approximating?

Restricting preimages to rectangular subdomains:



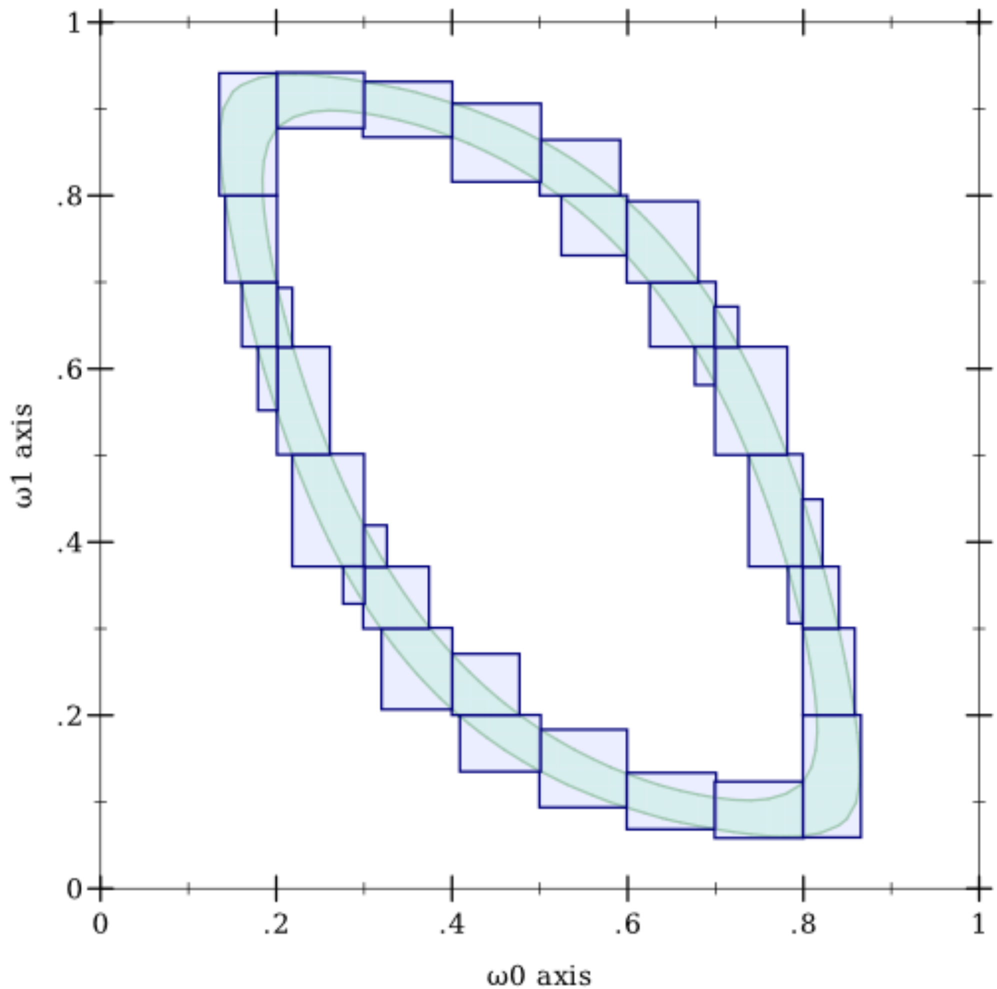
What About Approximating?

Restricting preimages to rectangular subdomains:



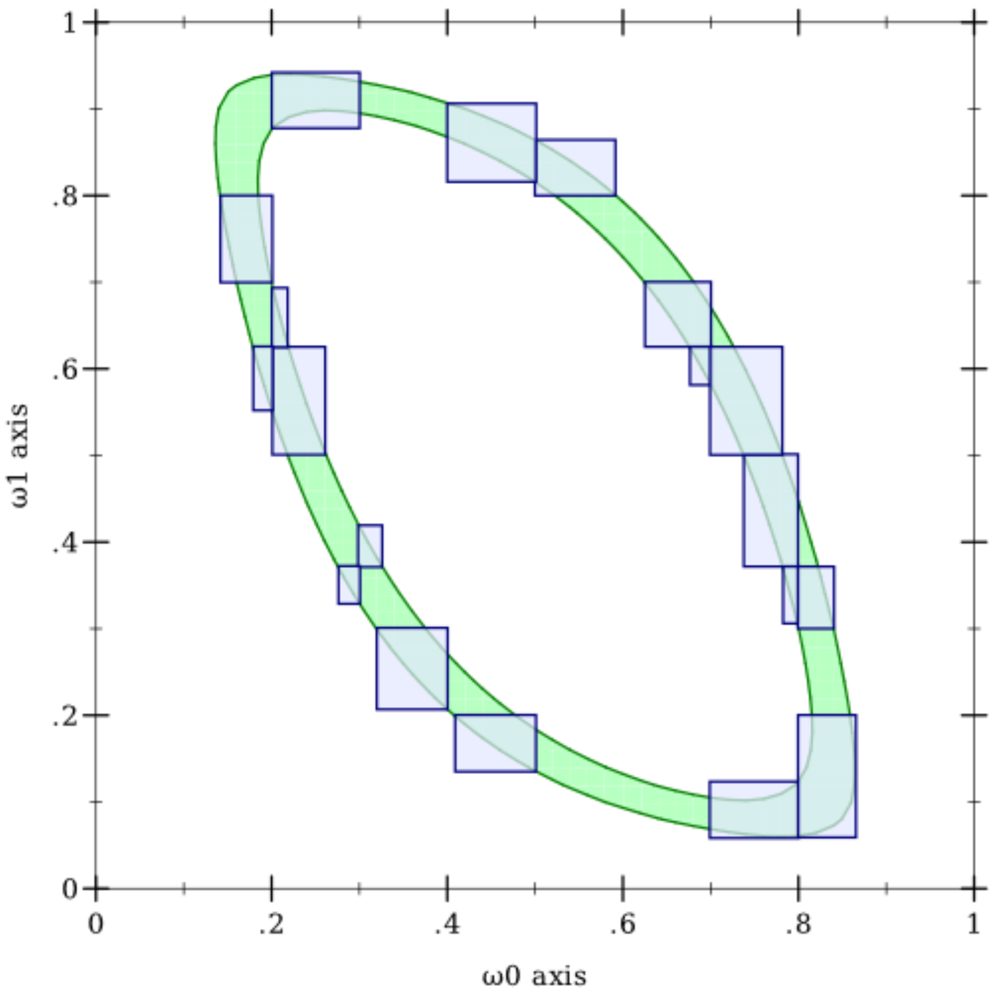
What About Approximating?

Restricting preimages to rectangular subdomains:



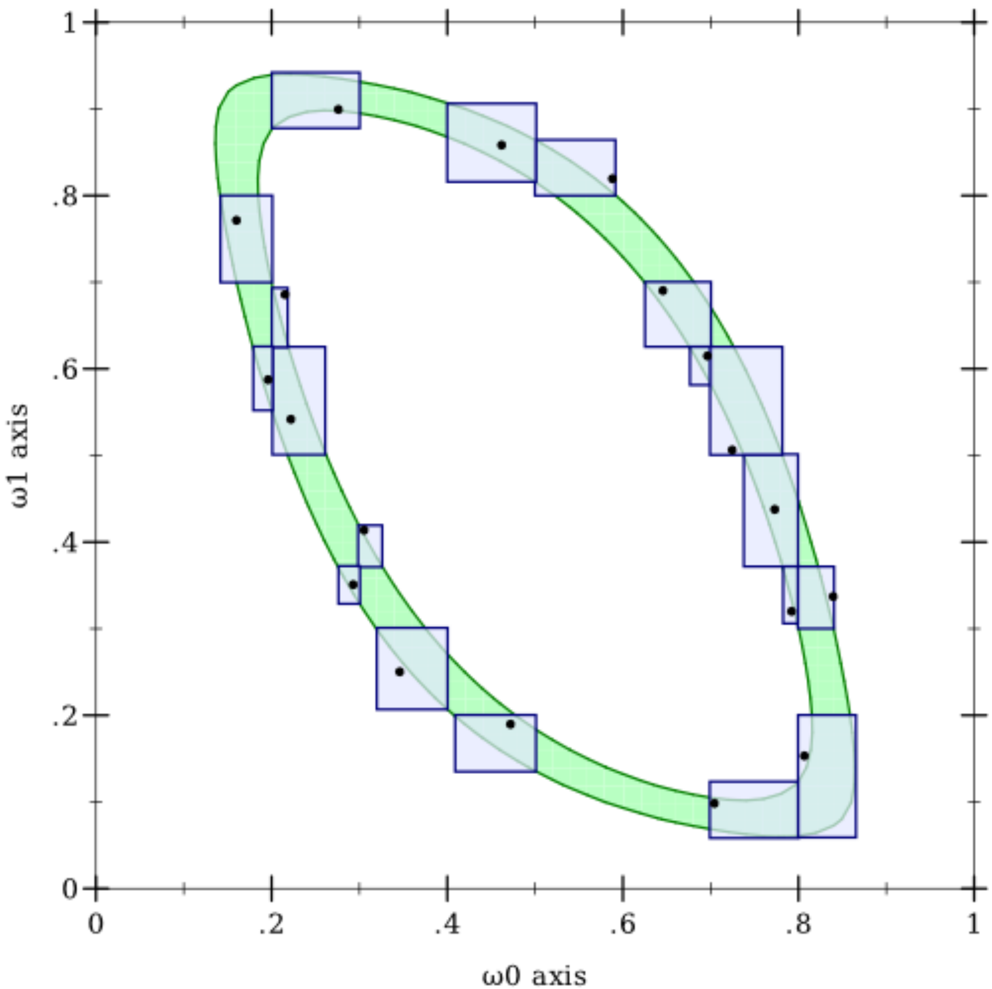
What About Approximating?

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:

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

Standard Interpretation

- Grammar:

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

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

Standard Interpretation

- Grammar:

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

- 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:

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

- 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 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 Semantics

- **Compositional**: every term's meaning depends only on its immediate subterms' meanings
- Advantage: proofs about all programs by structural induction
- Example: meaning of $\langle x + y, x \cdot y \rangle$

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 $\langle x + y, x \cdot y \rangle$

$$[\![\langle x + y, x \cdot y \rangle]\!] \;=\; \mathrm{pair}\ [\![x + y]\!]\ [\![x \cdot y]\!]$$

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 $\langle x + y, x \cdot y \rangle$

$$[\![\langle x + y, x \cdot y \rangle]\!] \;=\; \mathrm{pair}\ [\![x + y]\!]\ [\![x \cdot y]\!]$$

- Nonexample:

$$[\![\langle x + y, x \cdot y \rangle]\!] \;=\; \mathrm{pair}\ (\mathrm{plus}\ [\![x]\!]\ [\![y]\!])\ [\![x \cdot y]\!]$$

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 $\langle x + y, x \cdot y \rangle$

$$[\![\langle x + y, x \cdot y \rangle]\!] \ = \ \text{pair} \ [\![x + y]\!] \ [\![x \cdot y]\!]$$

- Nonexample:

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

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

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 $\langle x+y, x\cdot y\rangle$

$$[\![\langle x+y, x\cdot y\rangle]\!] \;=\; \text{pair}\ [\![x+y]\!]\ [\![x\cdot y]\!]$$

- Nonexample:

$$[\![\langle x+y, x\cdot y\rangle]\!] \;=\; \text{pair}\ (\text{plus}\ [\![x]\!]\ [\![y]\!])\ [\![x\cdot y]\!]$$

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

- 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

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

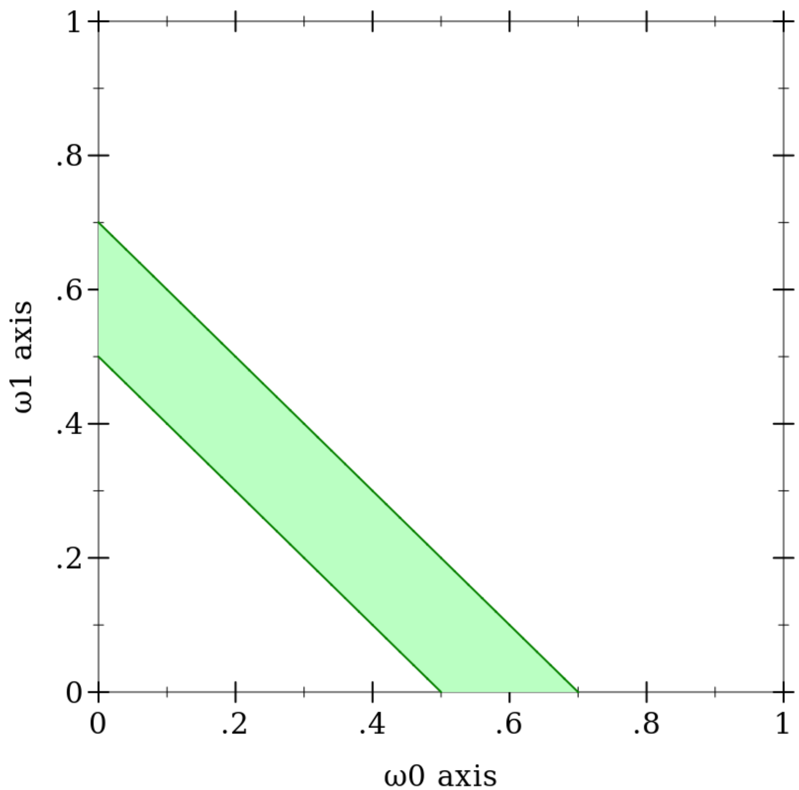
$$f \;=\; \text{pair}\ f_1\ f_2 \;=\; \lambda\,\omega.\,\langle \omega_0 + \omega_1, \omega_0 \cdot \omega_1 \rangle$$

Pair Preimages

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

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

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

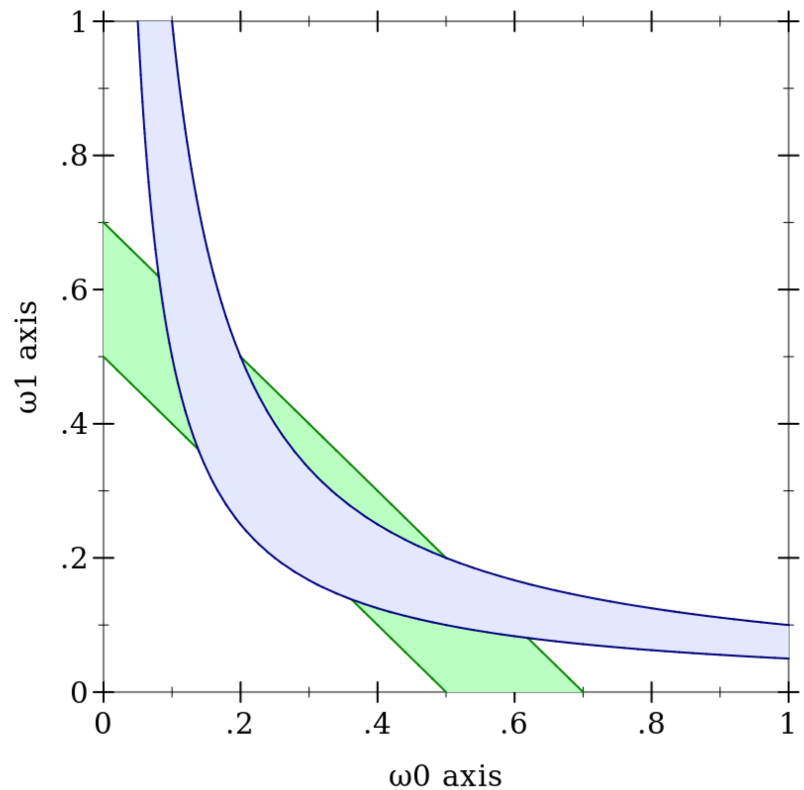


Pair Preimages

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

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

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

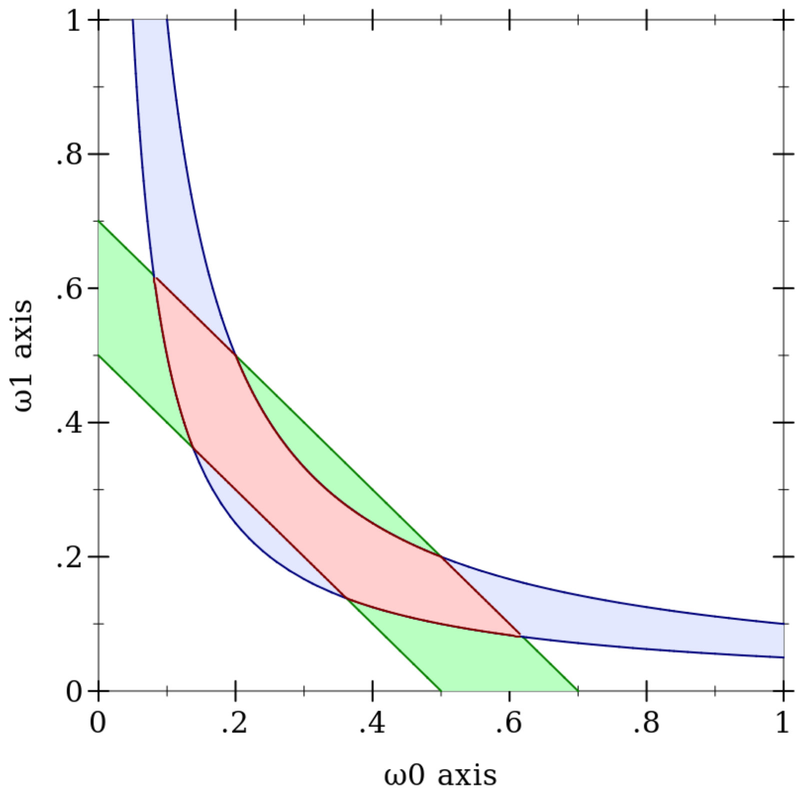


Pair Preimages

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

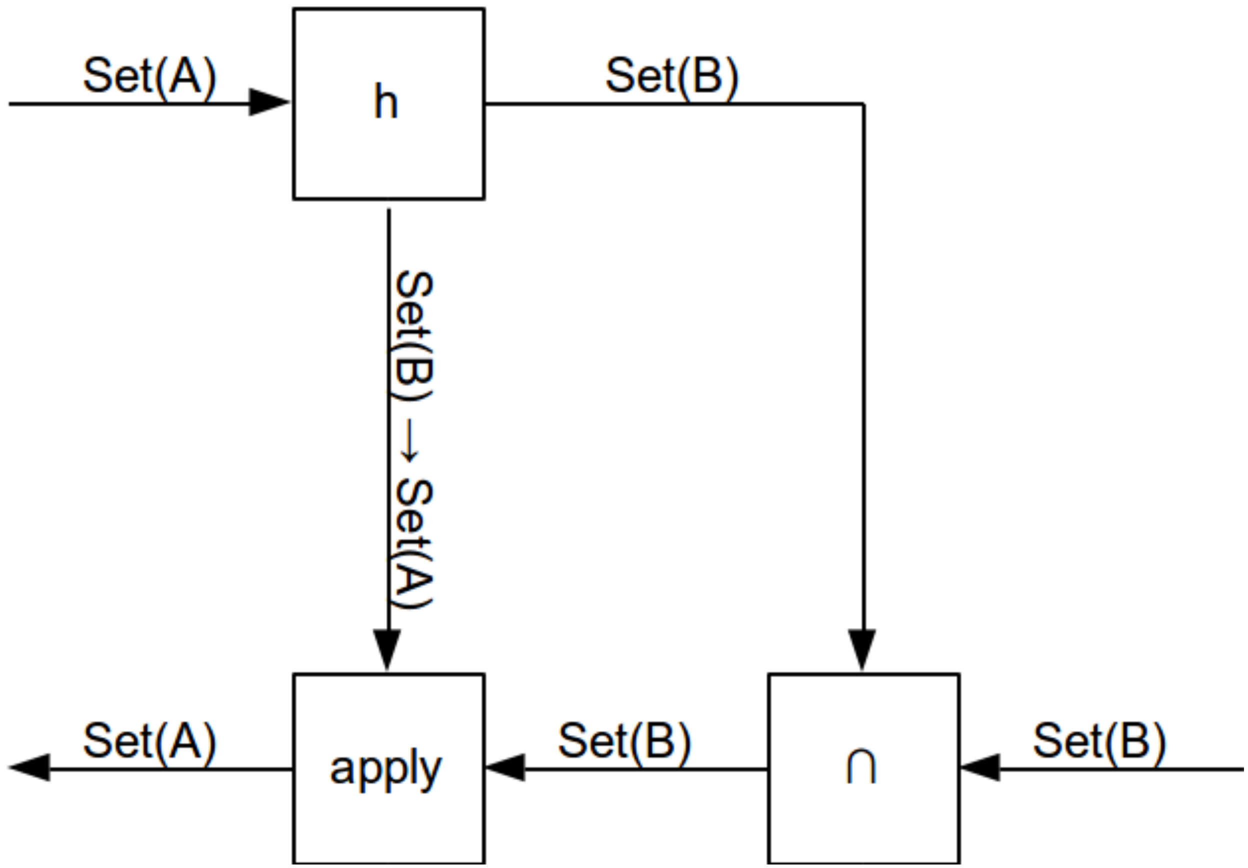
$$f = \text{pair } f_1 \; f_2 = \lambda\,\omega.\,\langle \omega_0 + \omega_1, \omega_0 \cdot \omega_1 \rangle$$

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



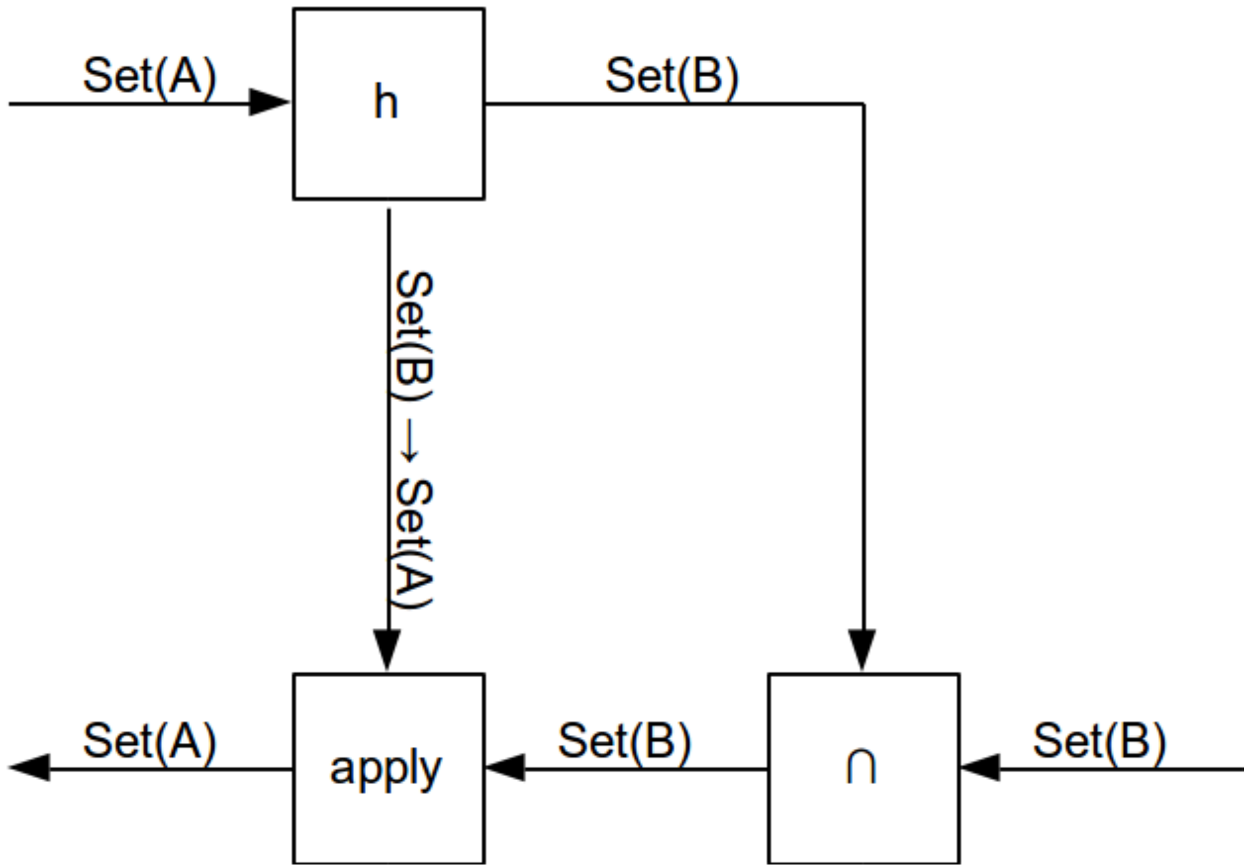
Nonstandard Interpretation: Computing Preimages

- Preimage computation:



Nonstandard Interpretation: Computing Preimages

- Preimage computation:



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

Nonstandard Interpretation: Preimages Under Pairing

- Pairing types:

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

Nonstandard Interpretation: Preimages Under Pairing

- Pairing types:

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

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

Theorem (correctness under pairing). If

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

Nonstandard Interpretation: Preimages Under Pairing

- Pairing types:

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

Theorem (correctness under pairing). If

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

Proof sketch. Preimages distribute over cartesian products.

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 $\mathrm{pair}\ f_1\ f_2$.

Proof sketch. Preimages distribute over cartesian products.

- Similar theorems for every kind of term

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 uncountable things?

Wait a Minute

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

Wait a Minute

- Q. Don't the interpretations of $[\![\cdot]\!]_{\mathrm{pre}}$ do uncountable 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 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 $[\![\cdot]\!]_{\mathrm{pre}}$ 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 $[\![\cdot]\!]_{\mathrm{pre}}$ 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.

Wait a Minute

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

Lambda-ZFC

λ calculus

Lambda-ZFC

+

λ calculus

Infinite sets and operations

Lambda-ZFC

+

=

λ calculus

Infinite sets and operations

λ_{ZFC}

Lambda-ZFC

+

=

λ calculus

Infinite sets and operations

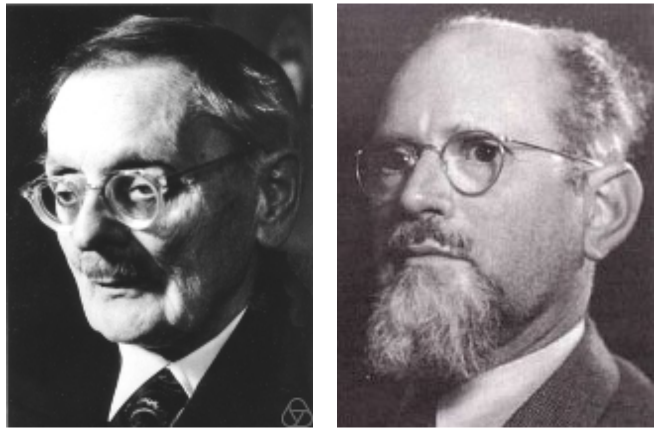
λ_{ZFC}

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

Lambda-ZFC

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

+

=

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 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:

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

Rectangular Approximation

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

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

- Define:

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

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:

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

- Define:

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

- 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). $[\![\cdot]\!]'_{\text{pre}}$ computes overapproximations of the preimages computed by $[\![\cdot]\!]_{\text{pre}}$.

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

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

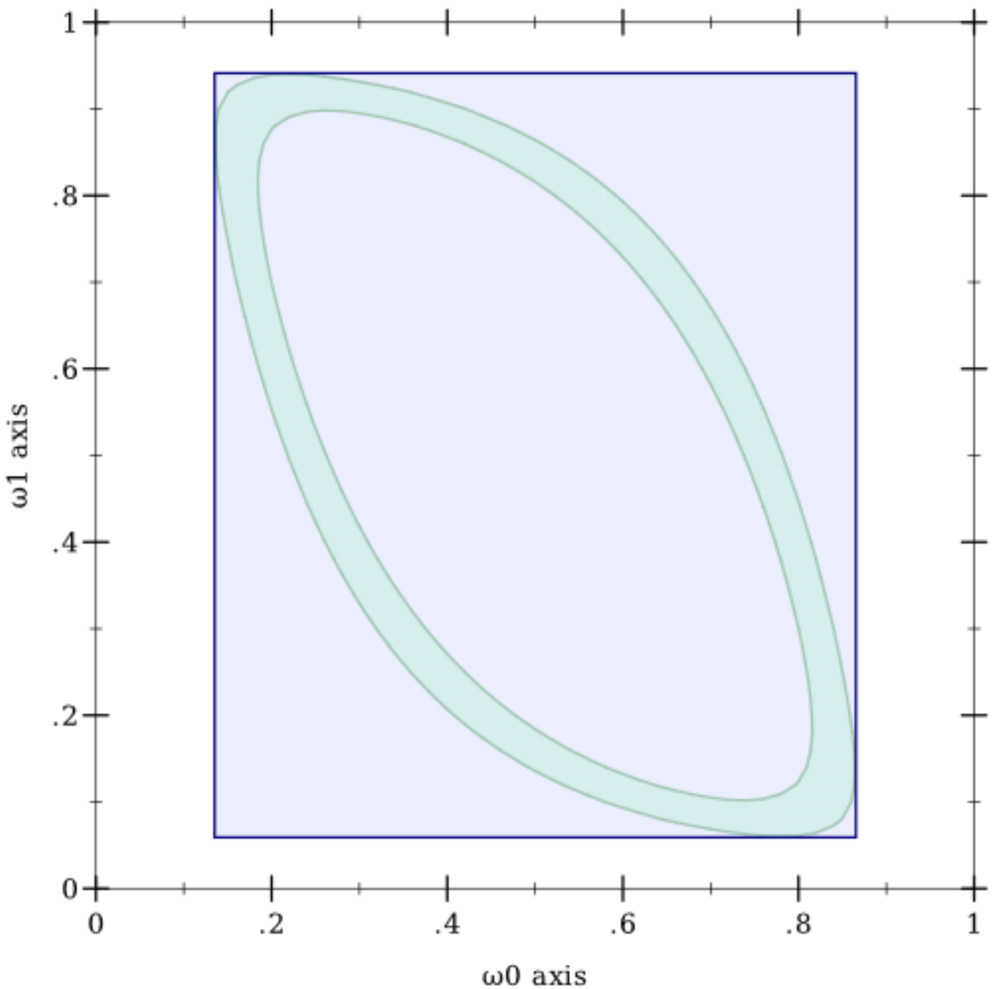
- Consequence: Partitioning the domain never increases approximate preimages

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

- Consequence: Refining preimage partitions never explodes

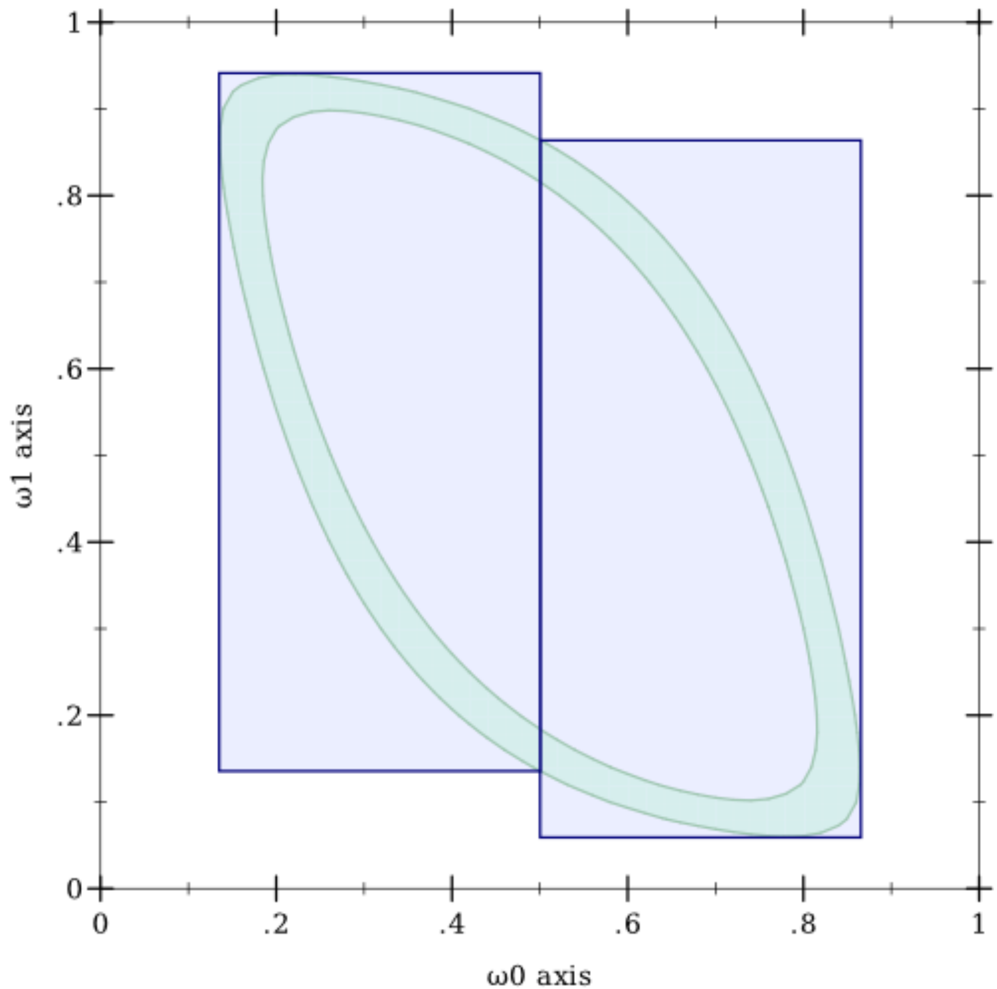
In Practice...

Theorems prove this always works:



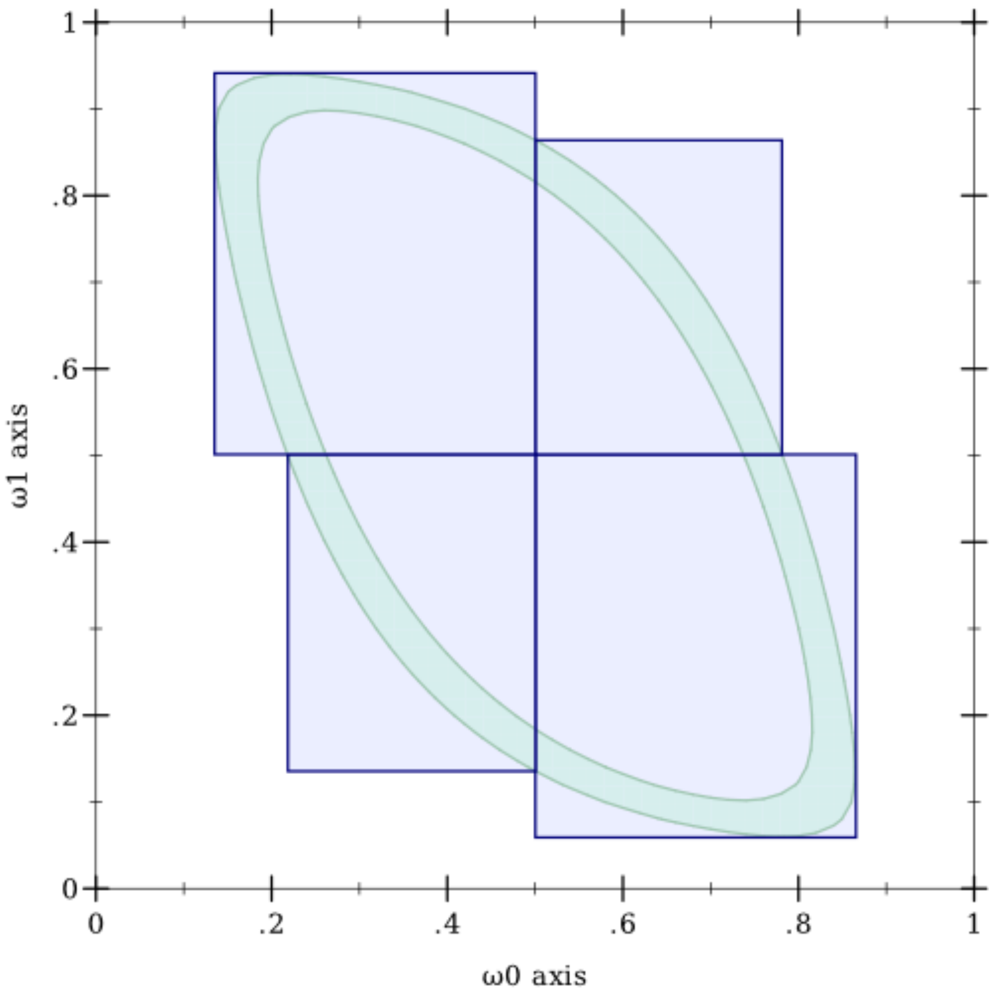
In Practice...

Theorems prove this always works:



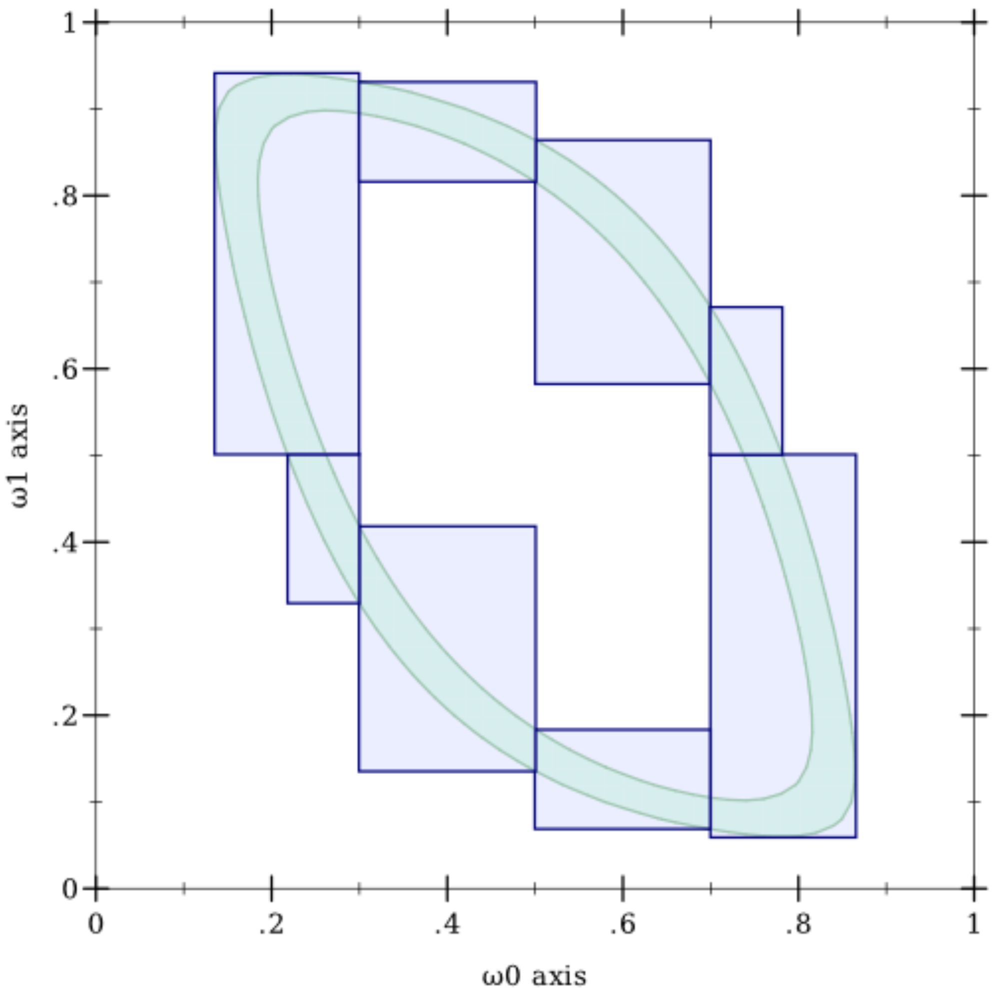
In Practice...

Theorems prove this always works:



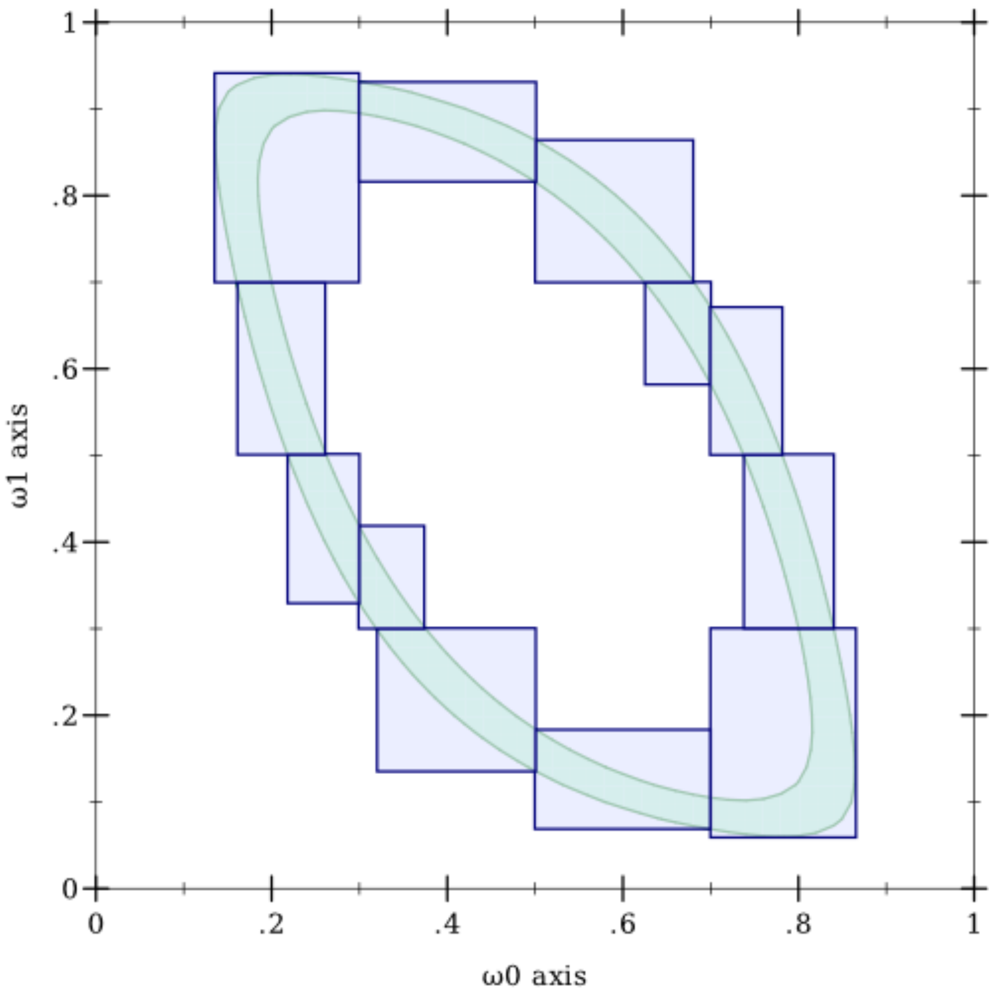
In Practice...

Theorems prove this always works:



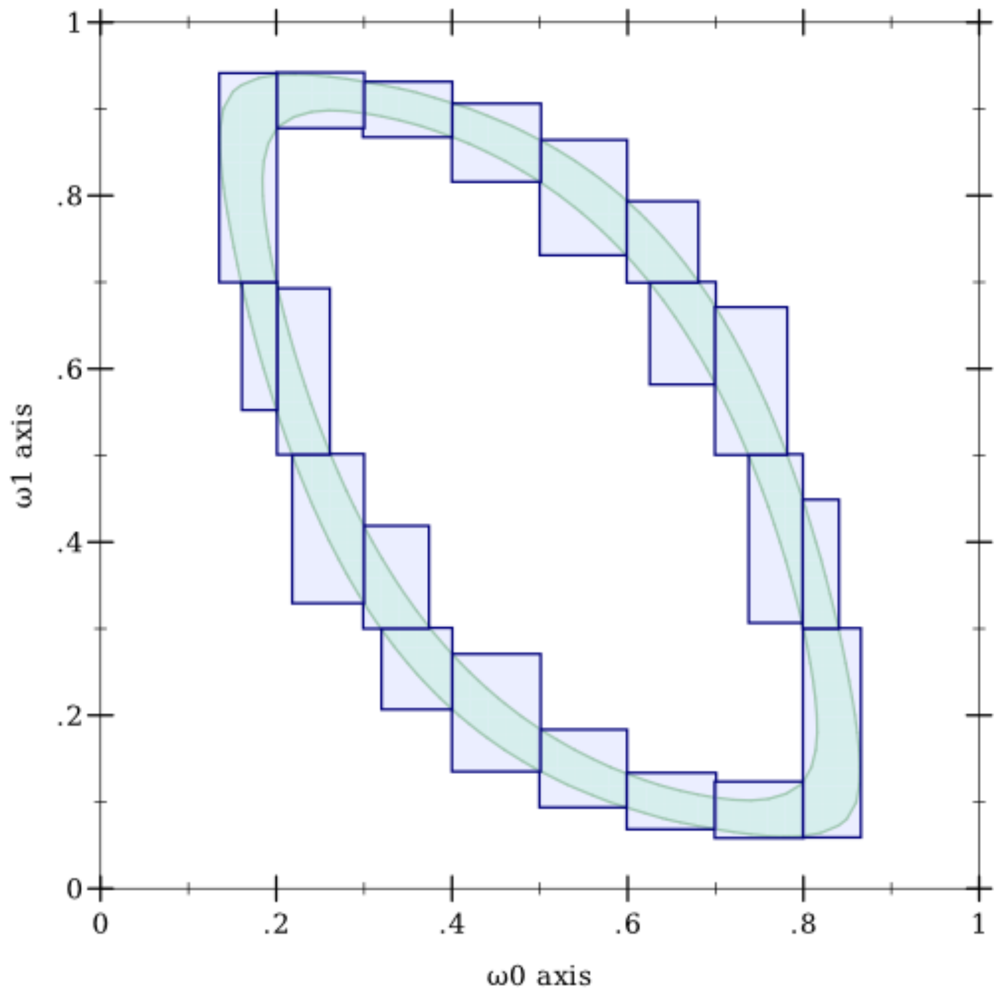
In Practice...

Theorems prove this always works:



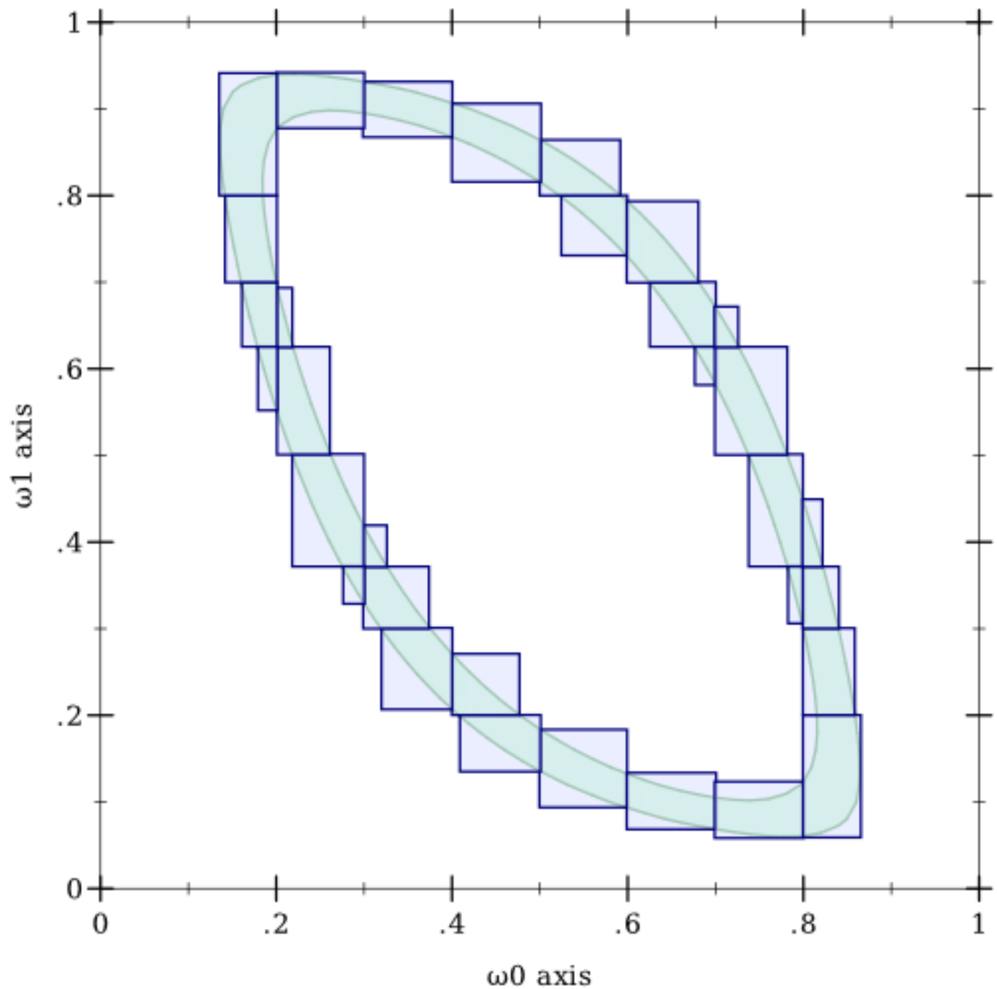
In Practice...

Theorems prove this always works:



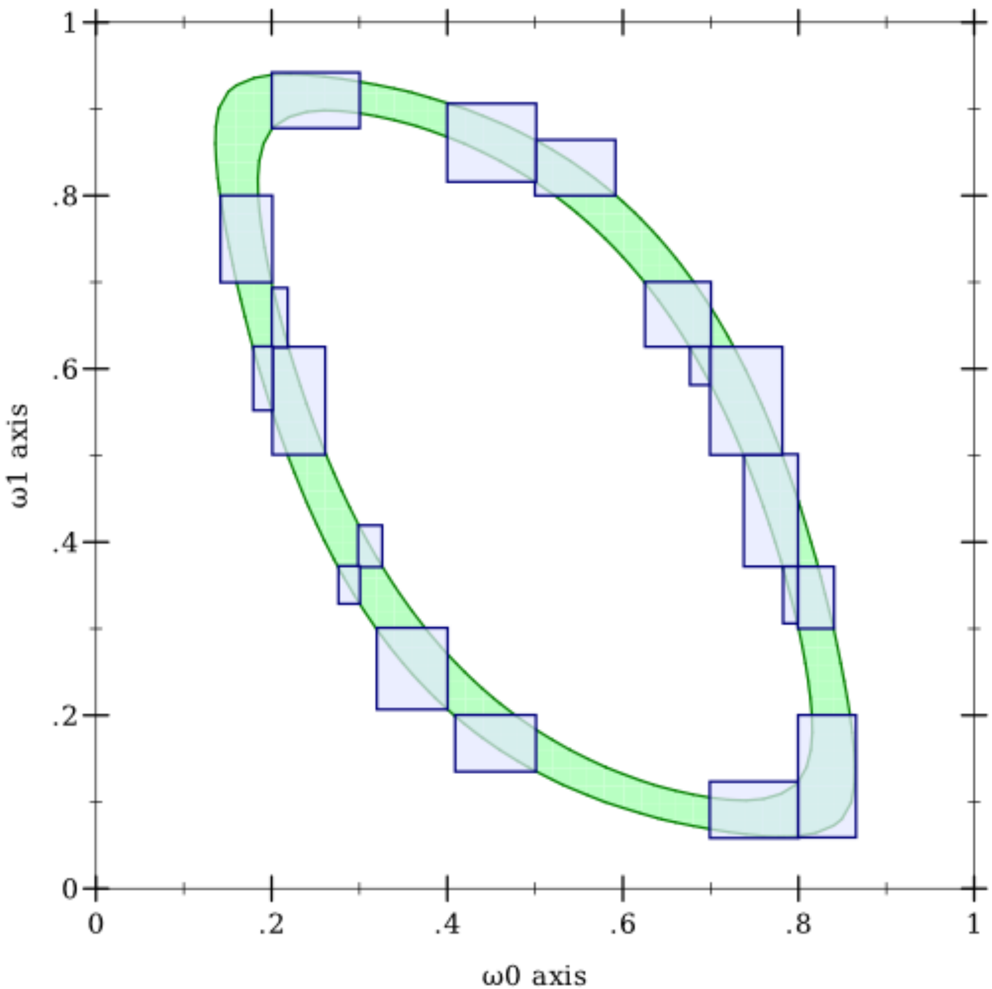
In Practice...

Theorems prove this always works:



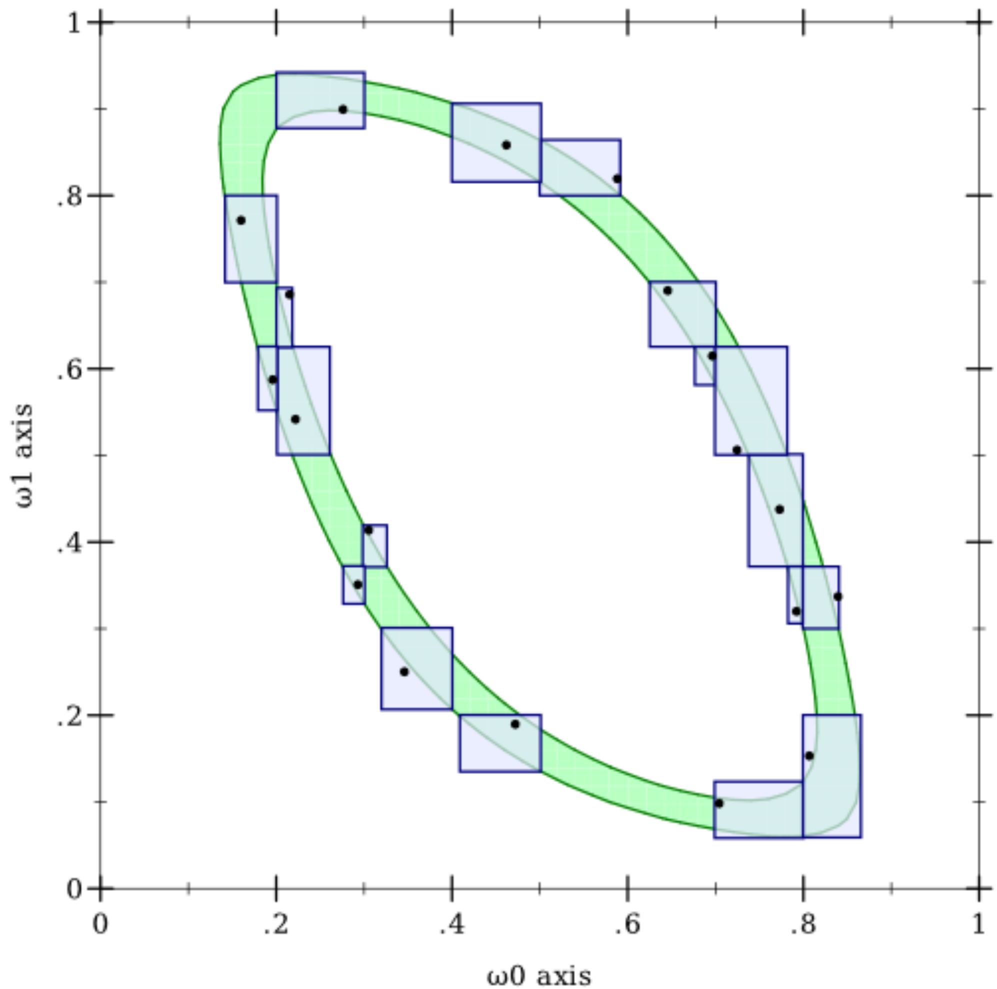
In Practice...

Theorems prove this always works:



In Practice...

Theorems prove this always works:



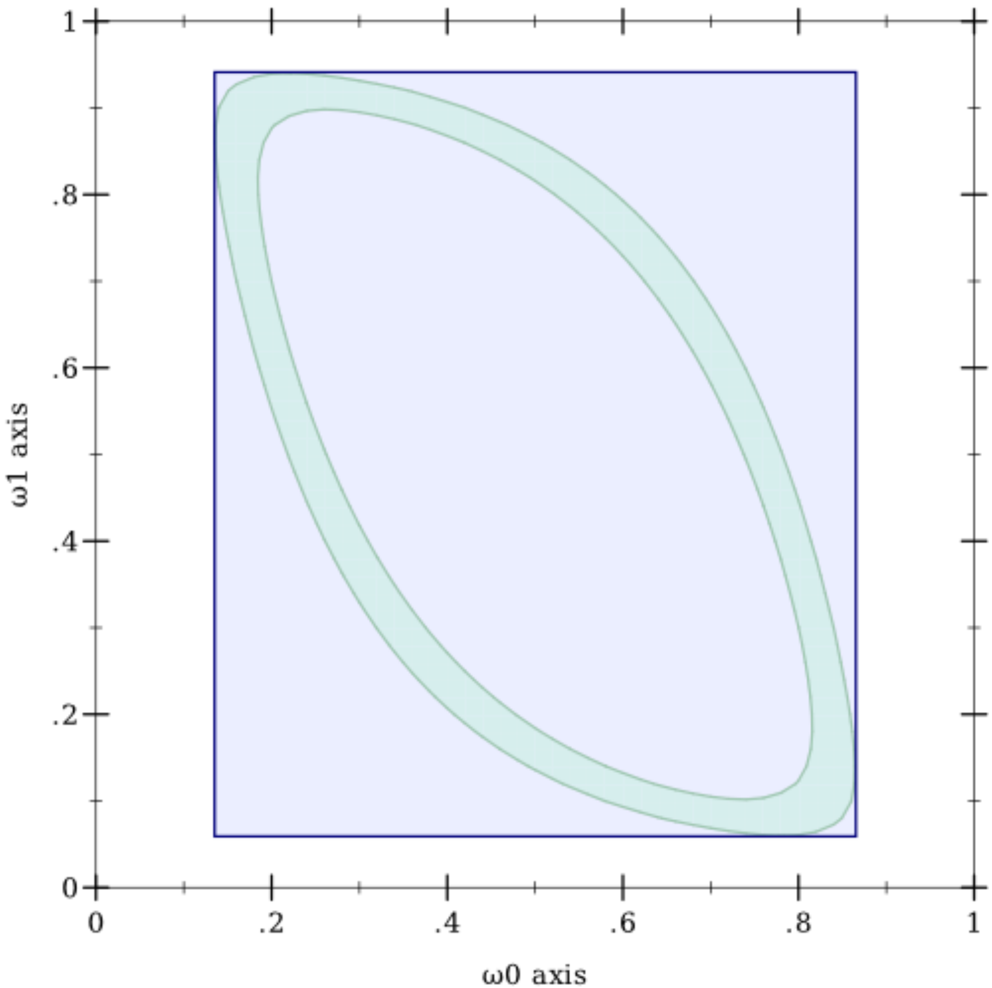
Importance Sampling

- Alternative to arbitrarily low-rate rejection sampling:

Importance Sampling

- Alternative to arbitrarily low-rate rejection sampling:

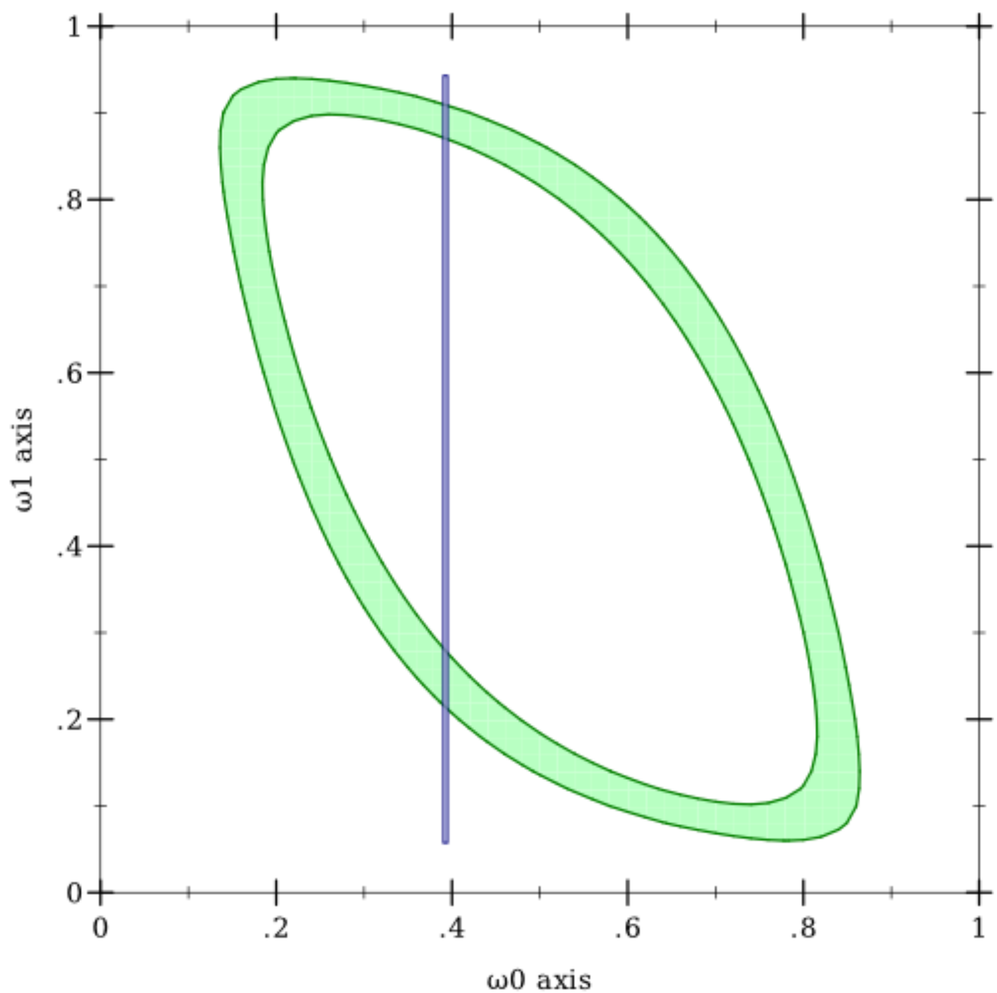
First, refine using preimage computation:



Importance Sampling

- Alternative to arbitrarily low-rate rejection sampling:

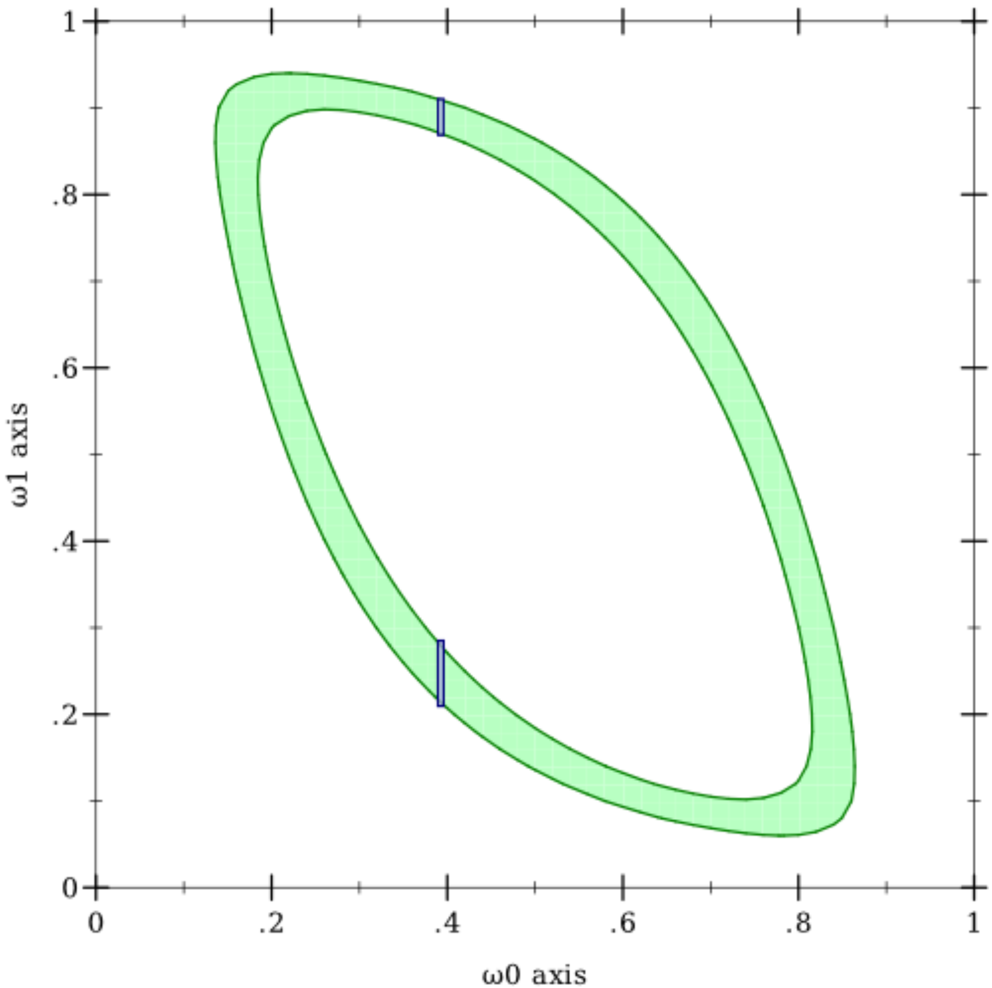
Second, randomly choose from arbitrarily fine partition:



Importance Sampling

- Alternative to arbitrarily low-rate rejection sampling:

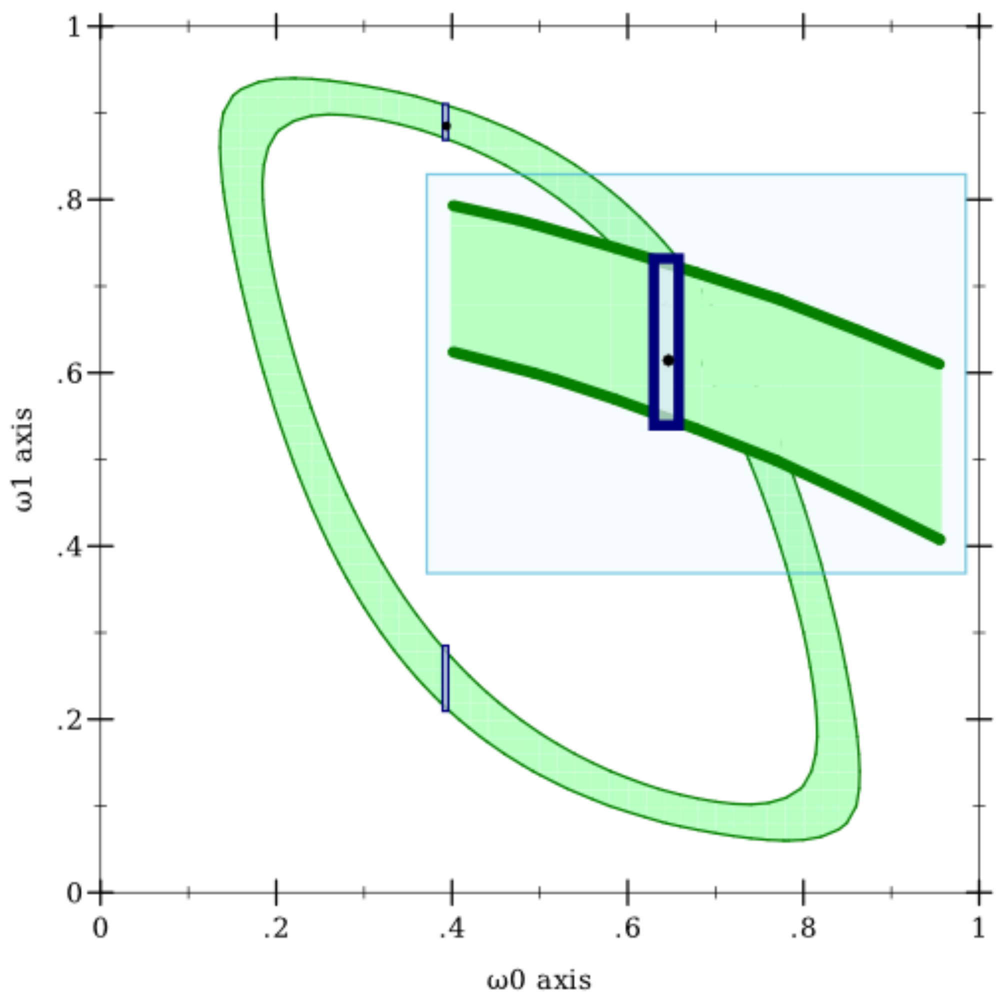
Third, refine again:



Importance Sampling

- Alternative to arbitrarily low-rate rejection sampling:

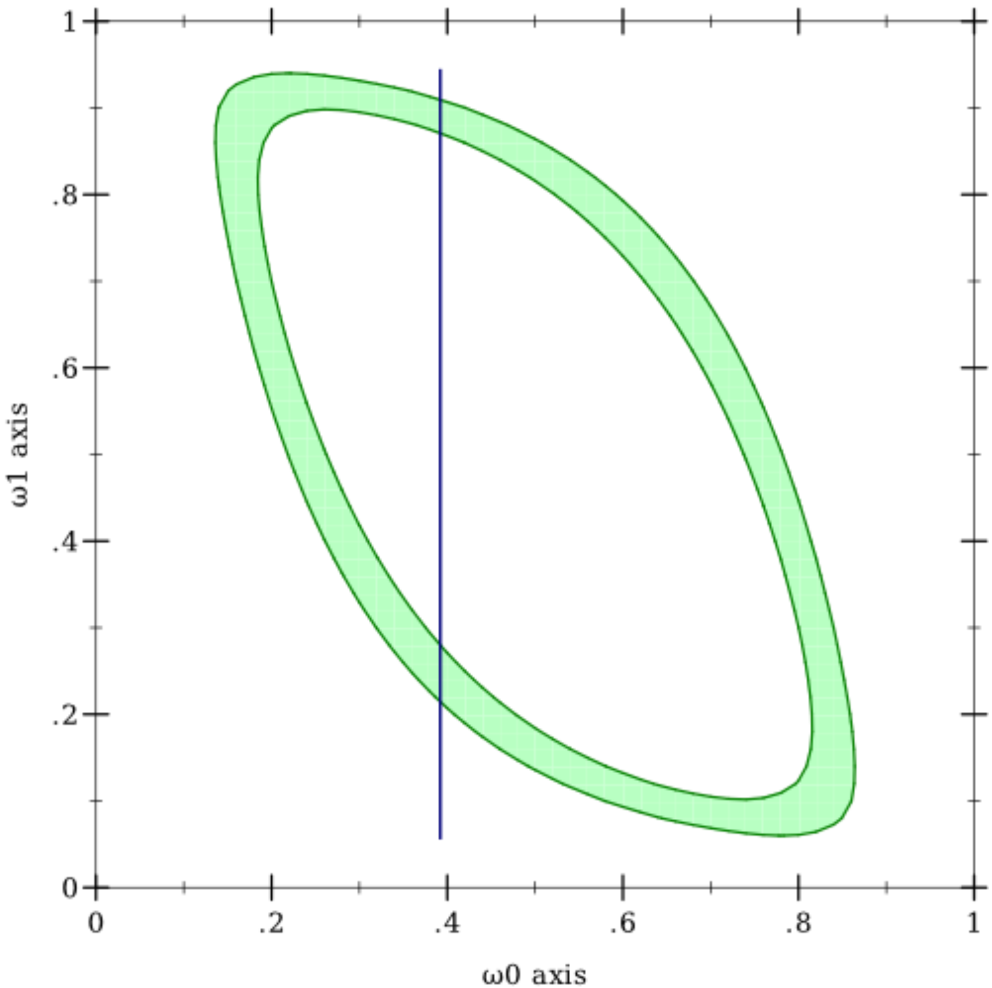
Fourth, sample uniformly:



Importance Sampling

- Alternative to arbitrarily low-rate rejection sampling:

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



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.

```
(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 ...)))))))
```

What About Recursion?

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

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

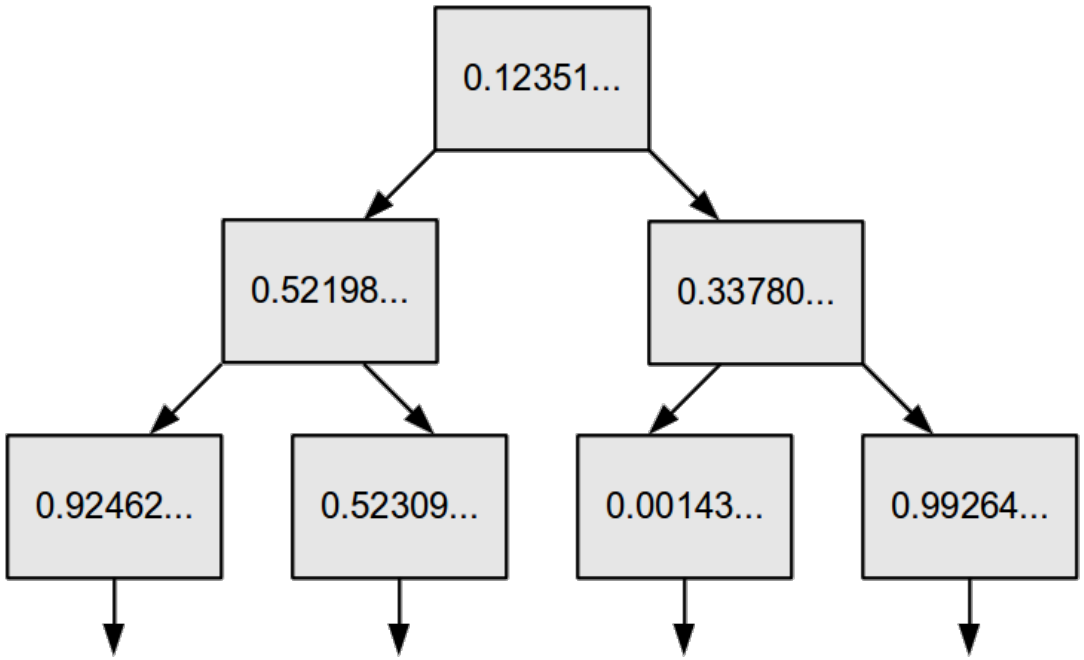
- Consider programs as being fully inlined (thus infinite):

```
(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

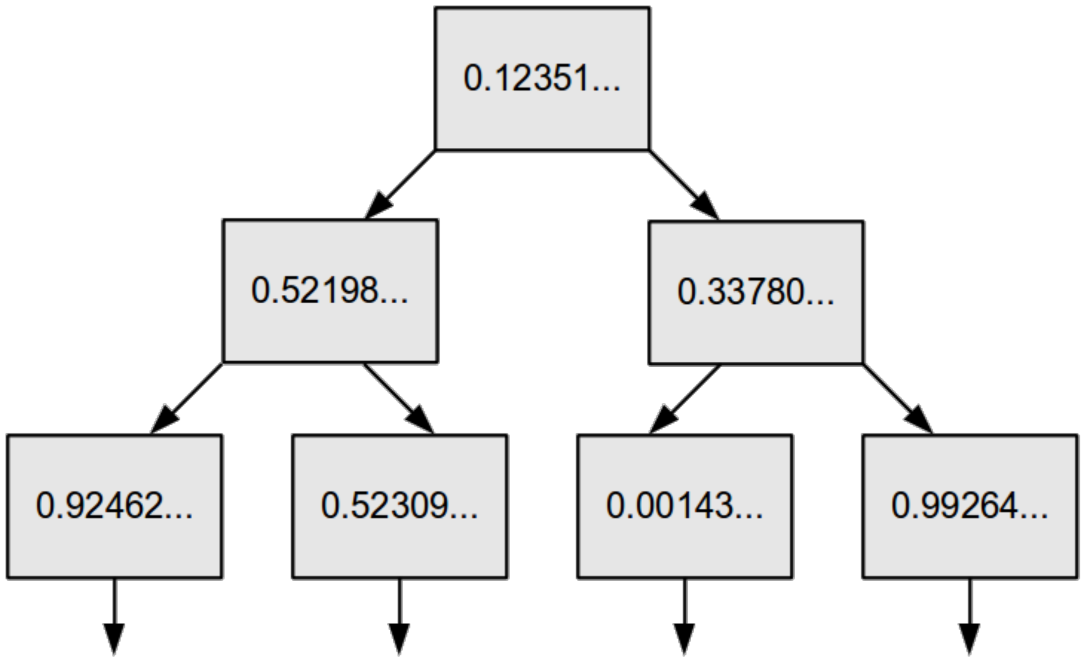
Program Domain Values

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



Program Domain Values

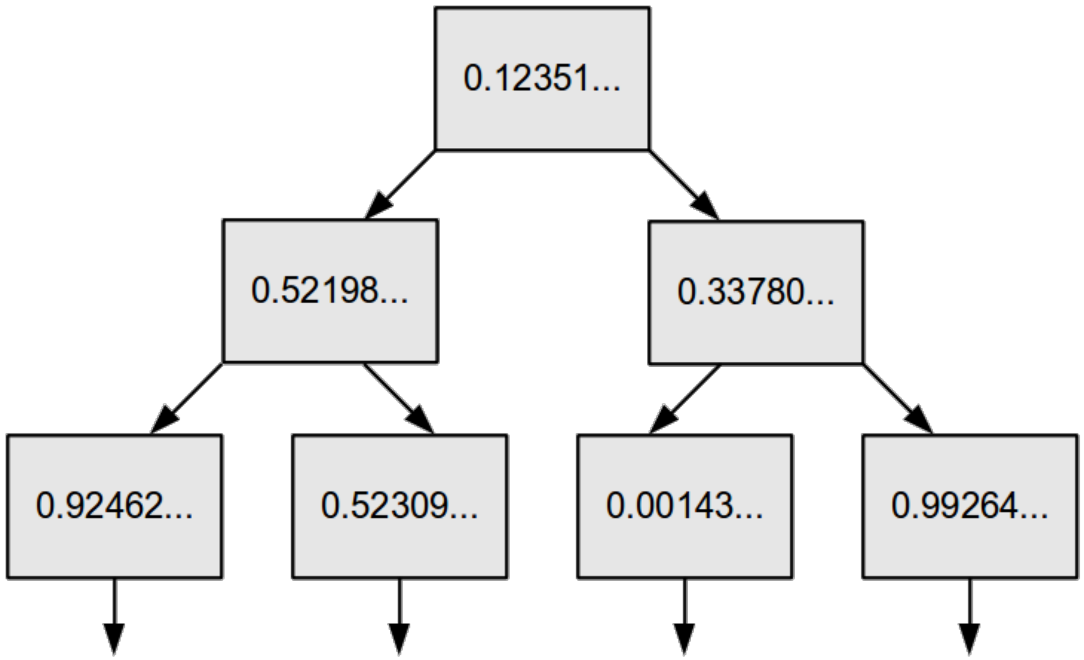
- Values $\omega \in \Omega$ are infinite binary trees:



- Every expression in a program is assigned a node

Program Domain Values

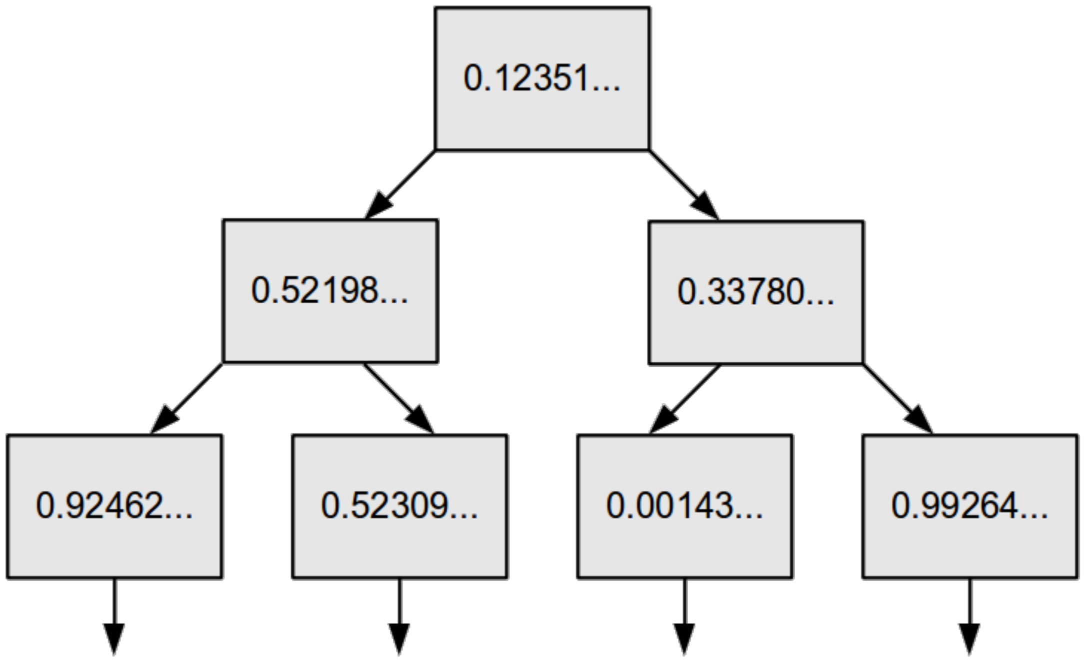
- Values $\omega \in \Omega$ are infinite binary trees:



- Every expression in a program is assigned a node
- Implemented using lazy trees of random values

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
- No probability *density* for domain, but there is a *measure*

Demo: Normal-Normal With Circular Condition

- Normal-Normal process:

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

Demo: Normal-Normal With Circular Condition

- Normal-Normal process:

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

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

Demo: Normal-Normal With Circular Condition

- Normal-Normal process:

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

- 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:

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

- 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))))))
```

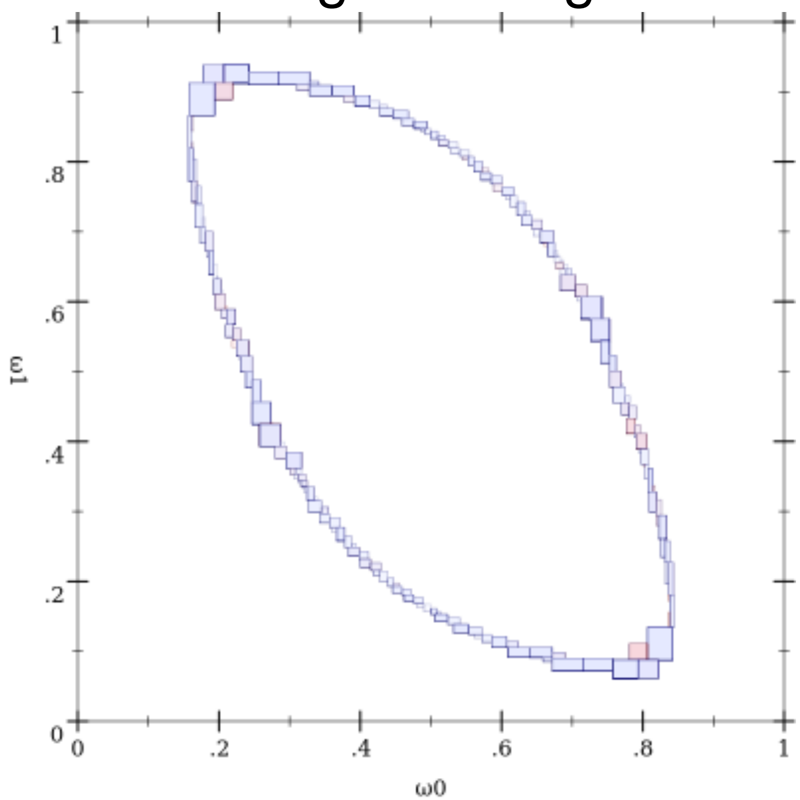
- Goal: Sample in the preimage of

```
(set-list reals reals (interval (- 1 ε) (+ 1 ε)))
```

Demo: Normal-Normal With Circular Condition

For ε = 0.01:

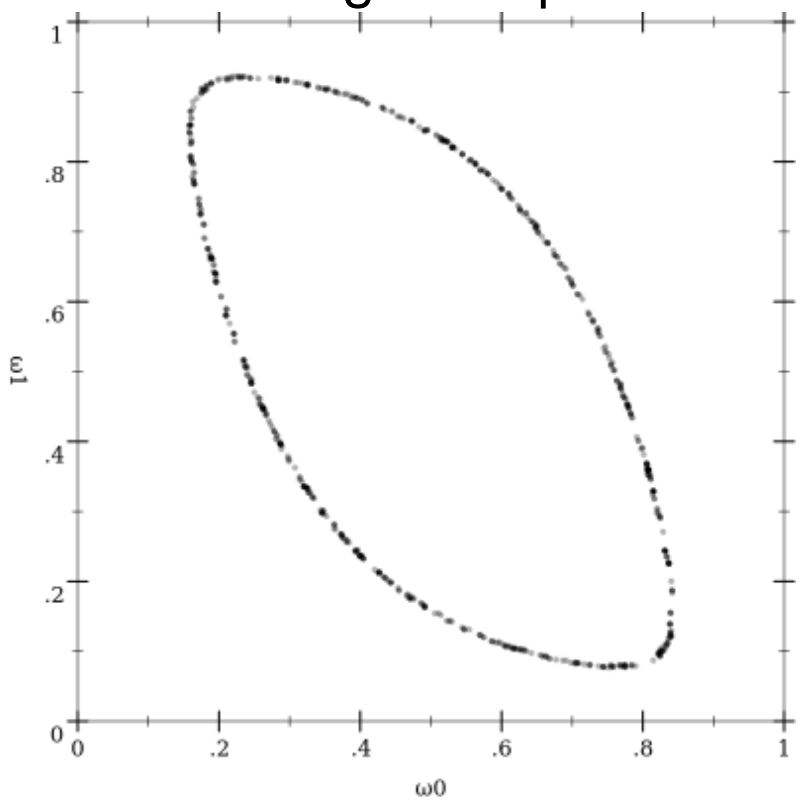
Preimage rectangles



Demo: Normal-Normal With Circular Condition

For $\varepsilon = 0.01$:

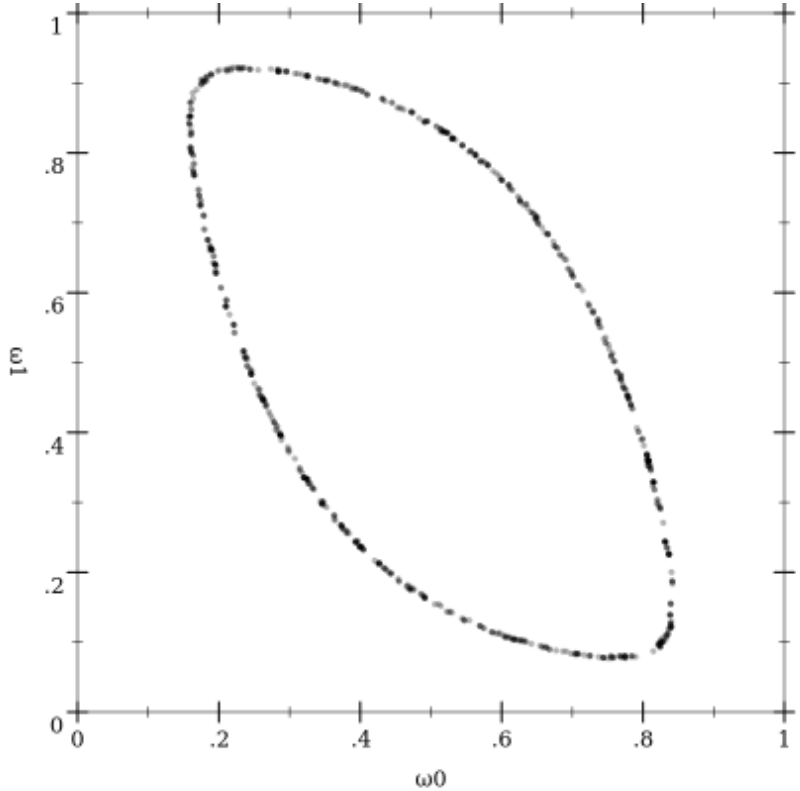
Preimage samples



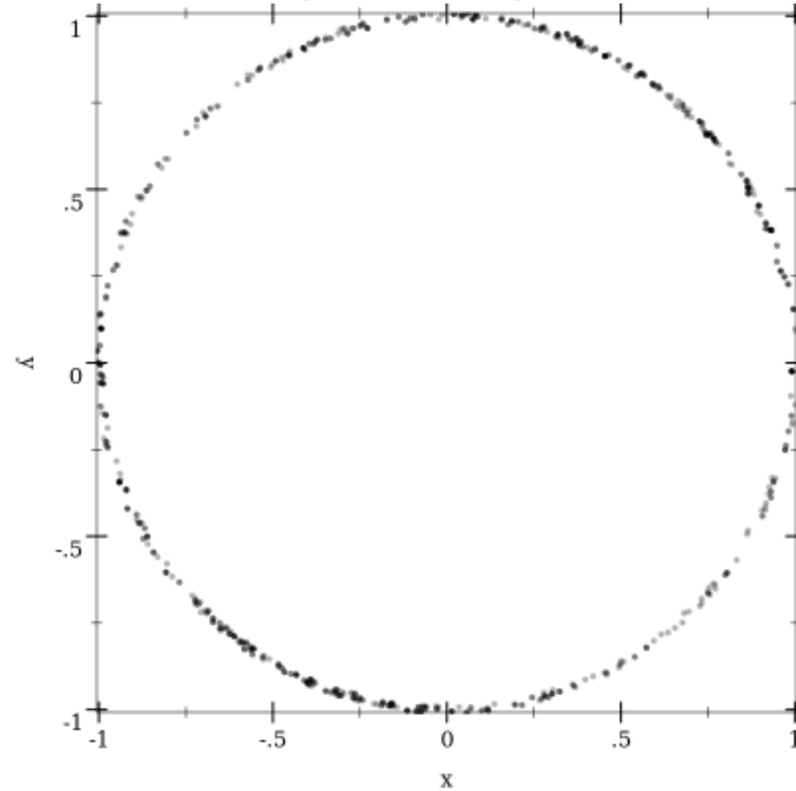
Demo: Normal-Normal With Circular Condition

For ε = 0.01:

Preimage samples



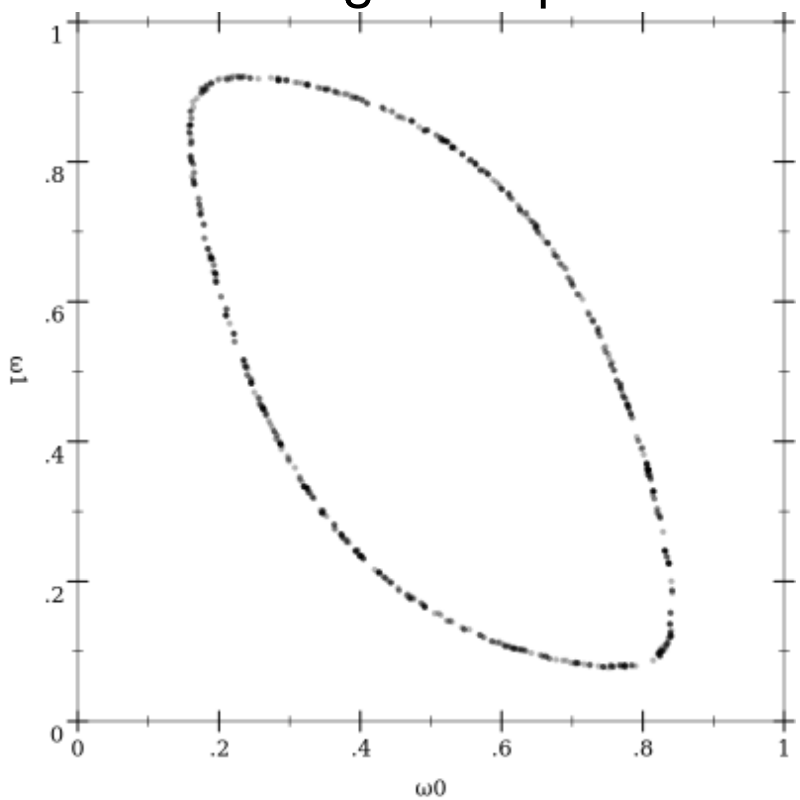
Output samples



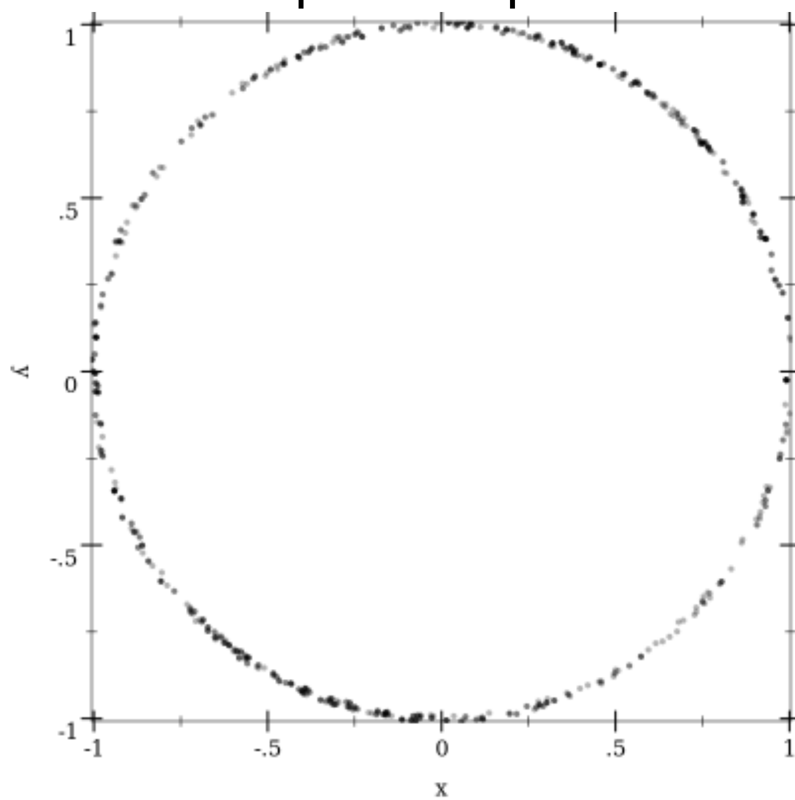
Demo: Normal-Normal With Circular Condition

For ε = 0.01:

Preimage samples



Output samples



- Works fine with much smaller ε

Demo: Thermometer

- Normal-Normal thermometer process:

$$
\begin{aligned}
X &\sim \mathrm{Normal}(90, 10) \\
Y &\sim \mathrm{Normal}(X, 1) \\
Y' &= \min(100, Y)
\end{aligned}
$$

Demo: Thermometer

- Normal-Normal thermometer process:

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

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

Demo: Thermometer

- Normal-Normal thermometer process:

$$
\begin{aligned}
X &\sim \mathrm{Normal}(90, 10) \\
Y &\sim \mathrm{Normal}(X, 1) \\
Y' &= \min(100, Y)
\end{aligned}
$$

- 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:

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

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

- Implementation:

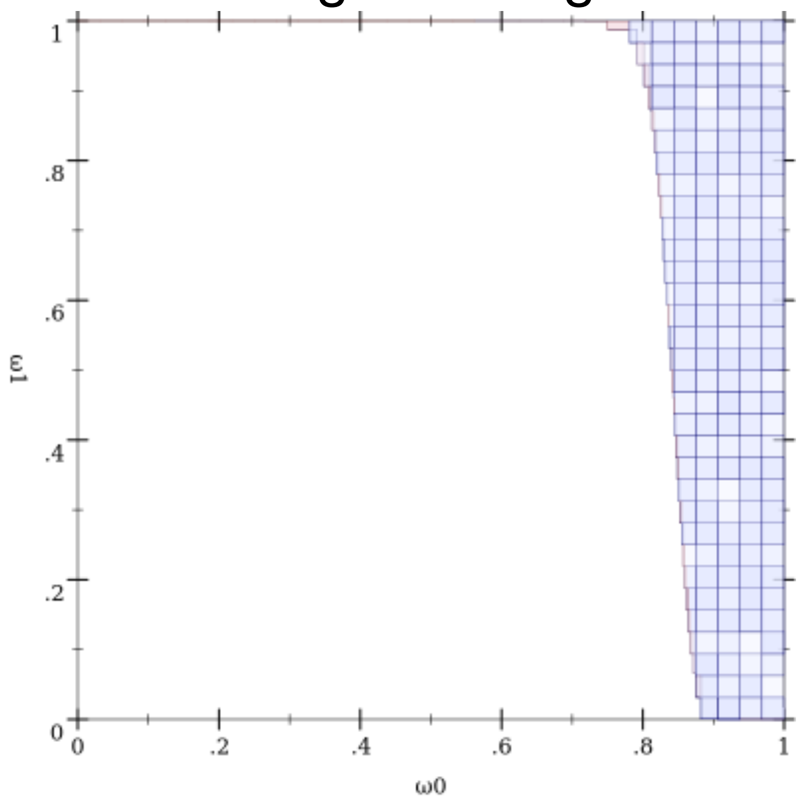
```
(define/drbayes e
  (let* ([x  (normal 90 10)]
         [y  (normal x 1)])
    (list x (if (> y 100) 100 y))))
```

- Goal: Sample in the preimage of

```
(set-list reals (interval 100 100))
```

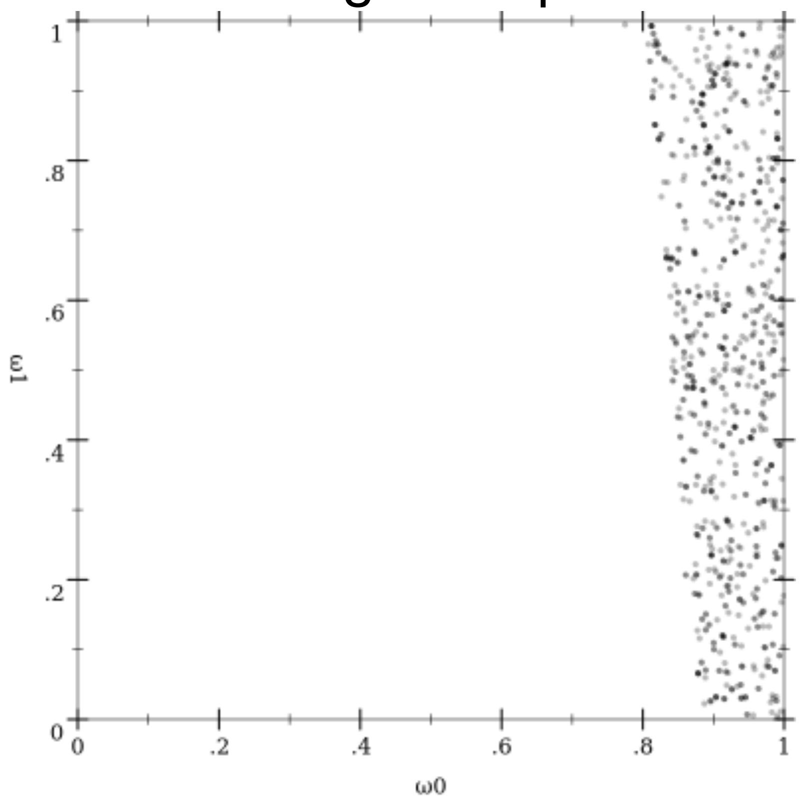
Demo: Thermometer

Preimage rectangles



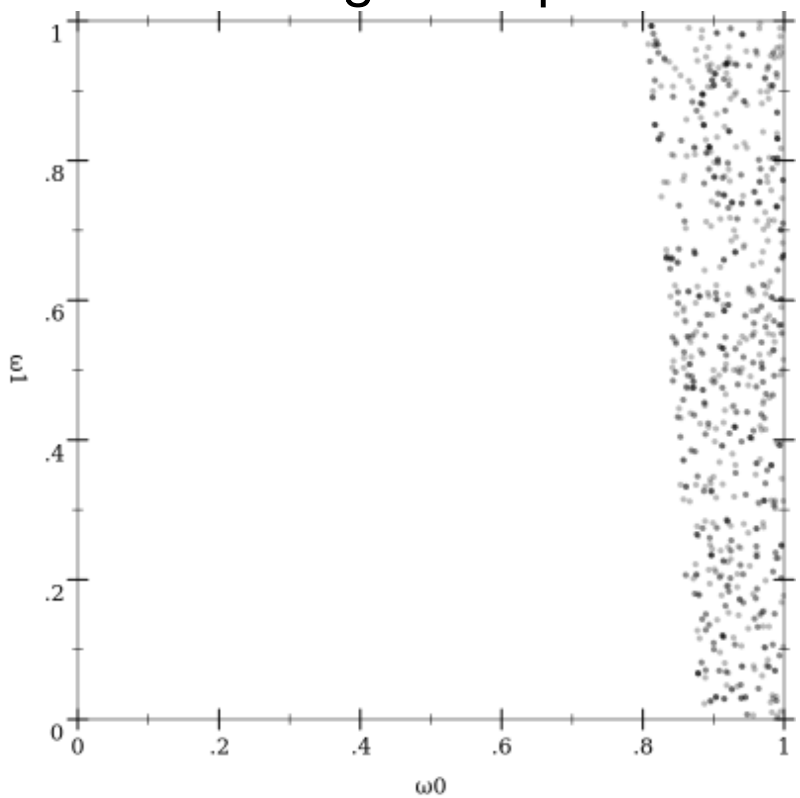
Demo: Thermometer

Preimage samples

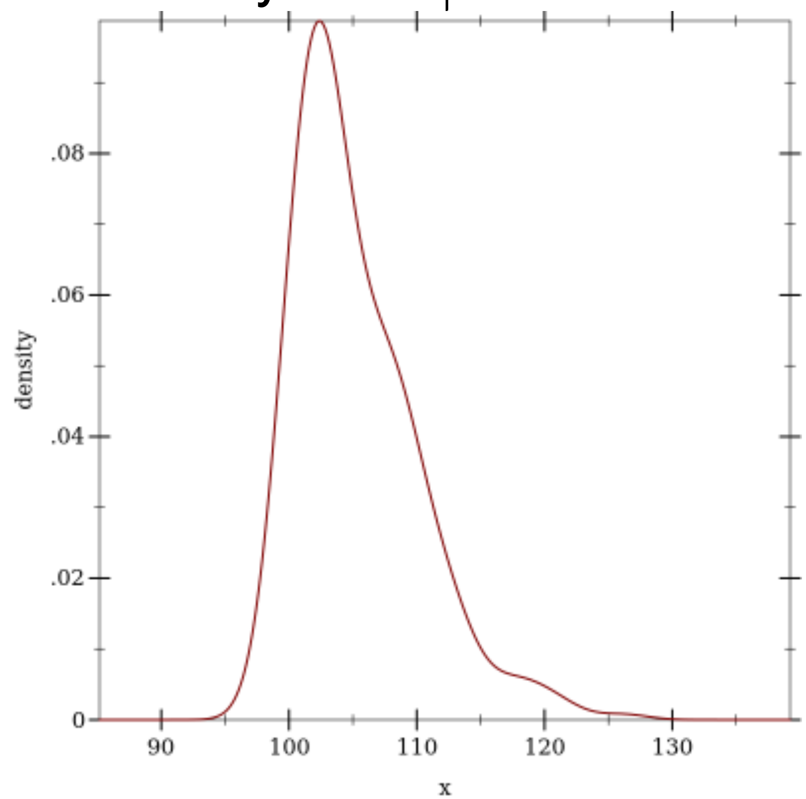


Demo: Thermometer

Preimage samples

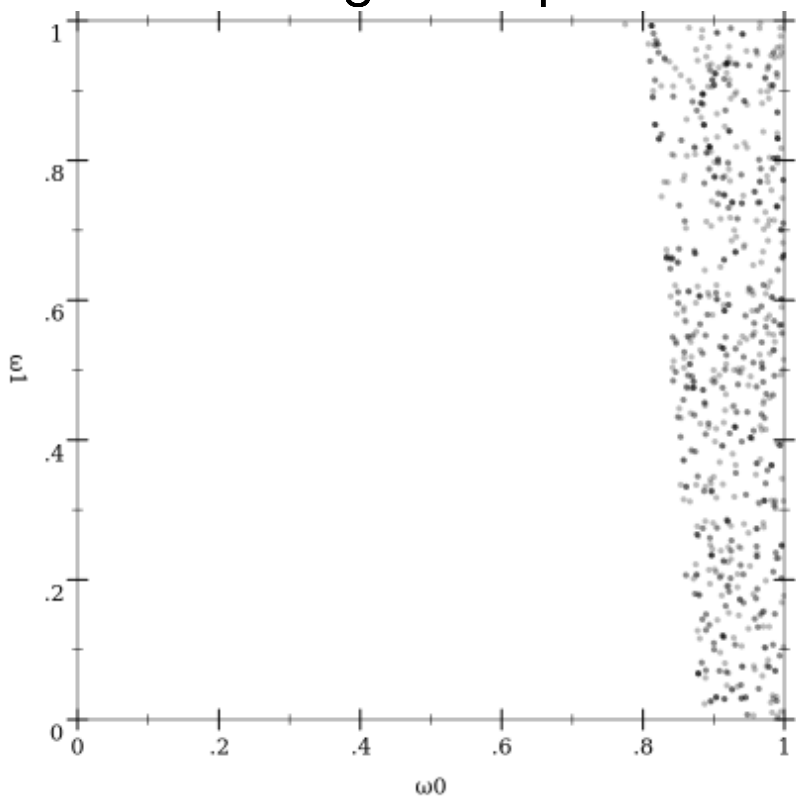


Density of $X \mid Y' = 100$

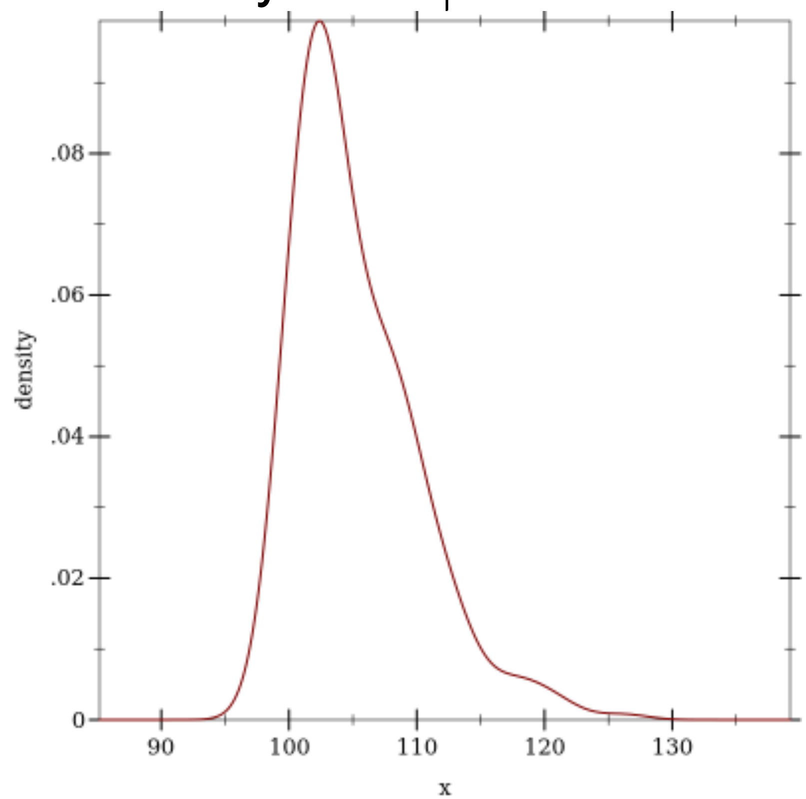


Demo: Thermometer

Preimage samples



Density of $X \mid Y' = 100$



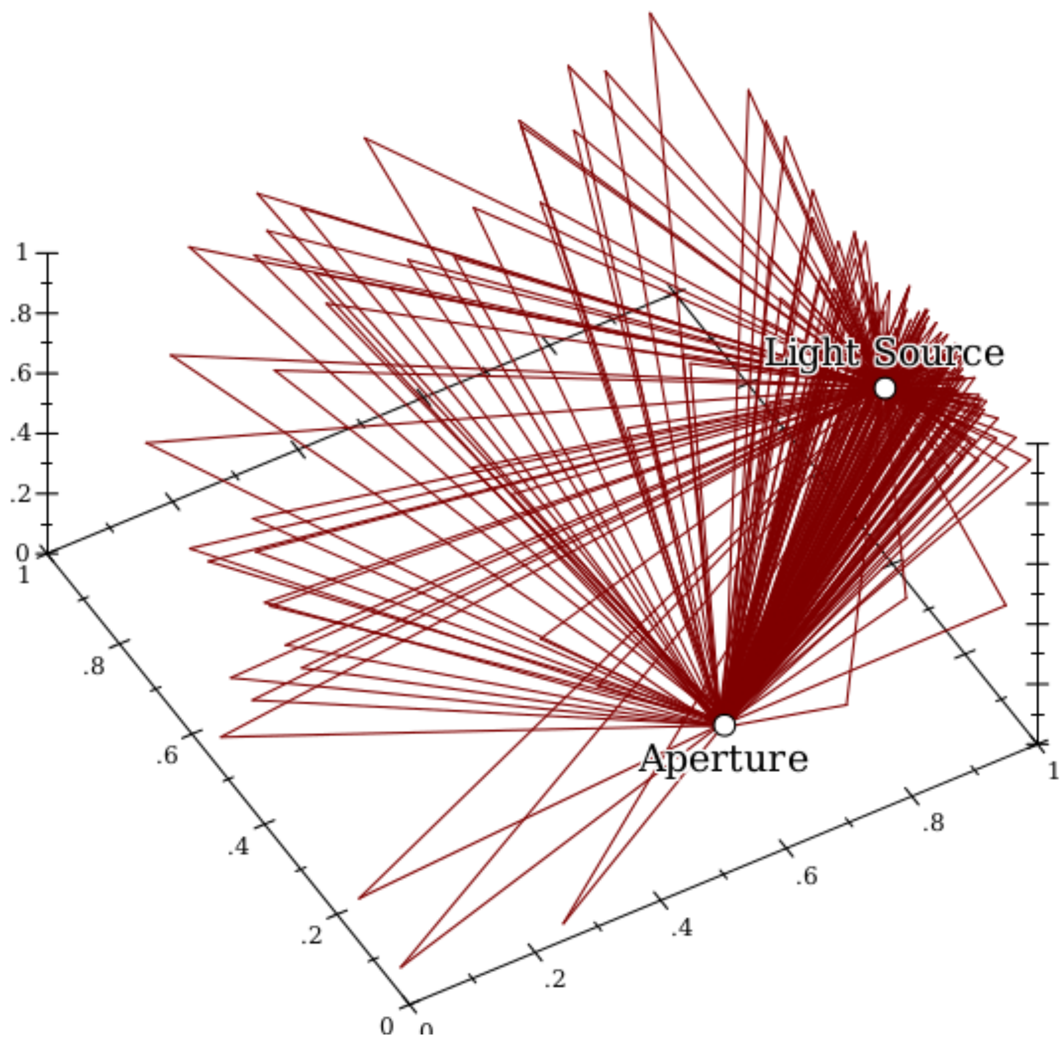
Calculated from samples: mean 105.1, stddev 4.6

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):

```
(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)))
```

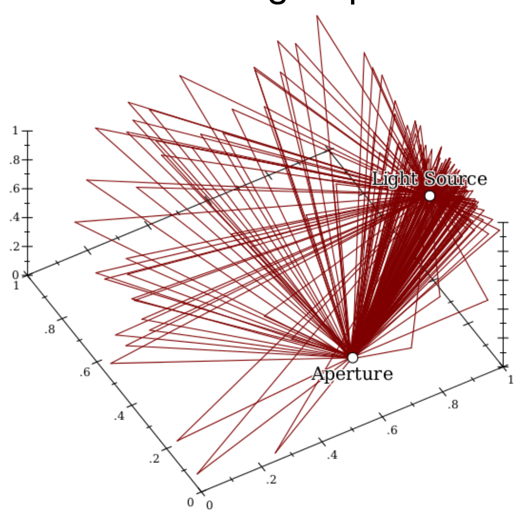
Demo: Stochastic Ray Tracing

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

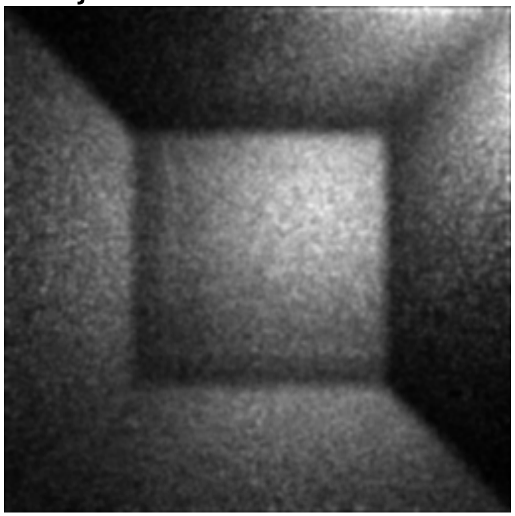
```
(define/drbayes (ray-plane-intersect p0 v n d)
  (let ([denom  (- (vec-dot v n))])
    (if (positive? denom)
        (let ([t  (/ (+ d (vec-dot p0 n)) denom)])
          (if (positive? t)
              (collision t (vec+ p0 (vec-scale v t)) n)
              #f))
        #f)))
```

- Constrained light path outputs:

Paths Through Aperture



Projected and Accumulated



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 < 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.

Thesis Statement

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

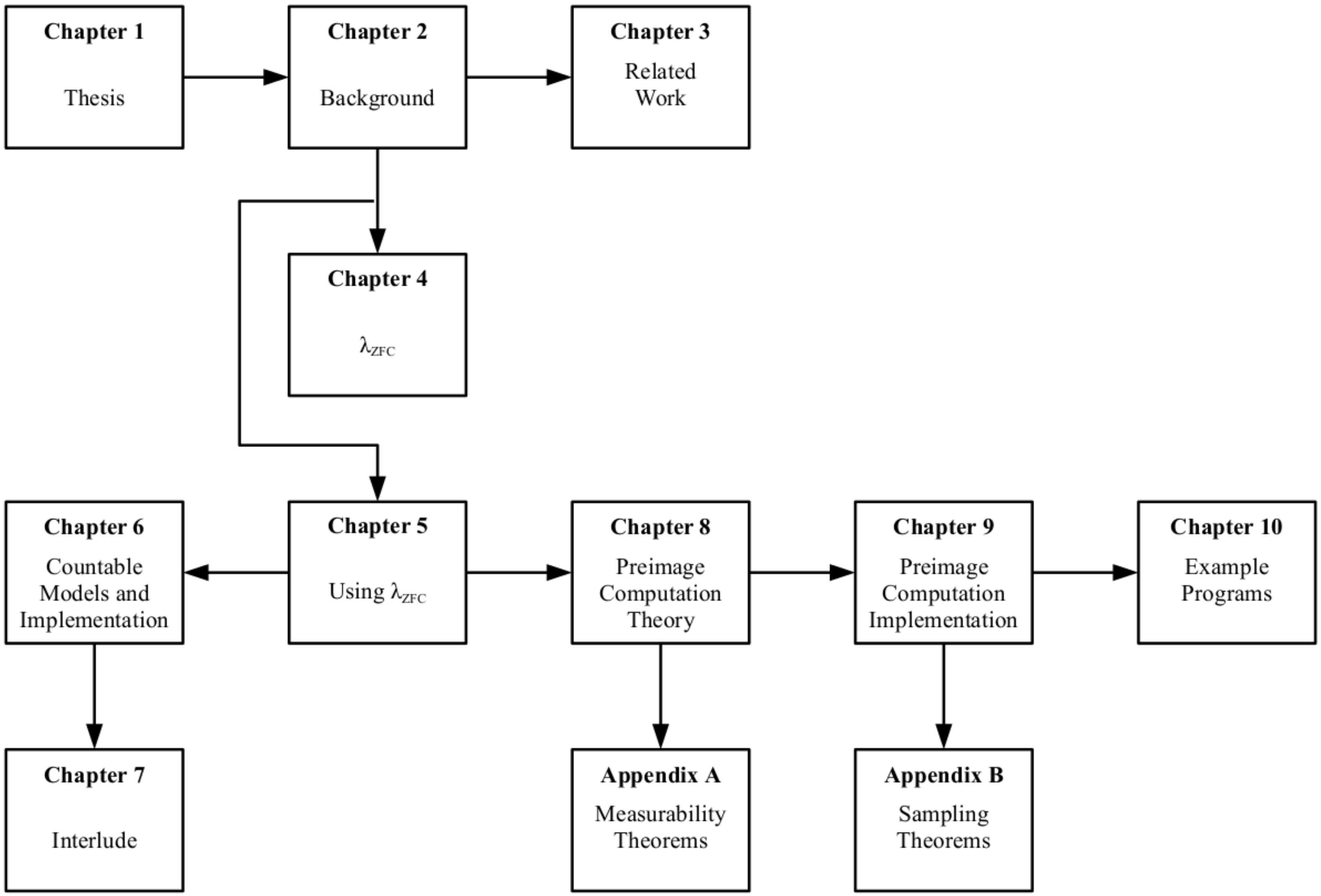
Measurability

- 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

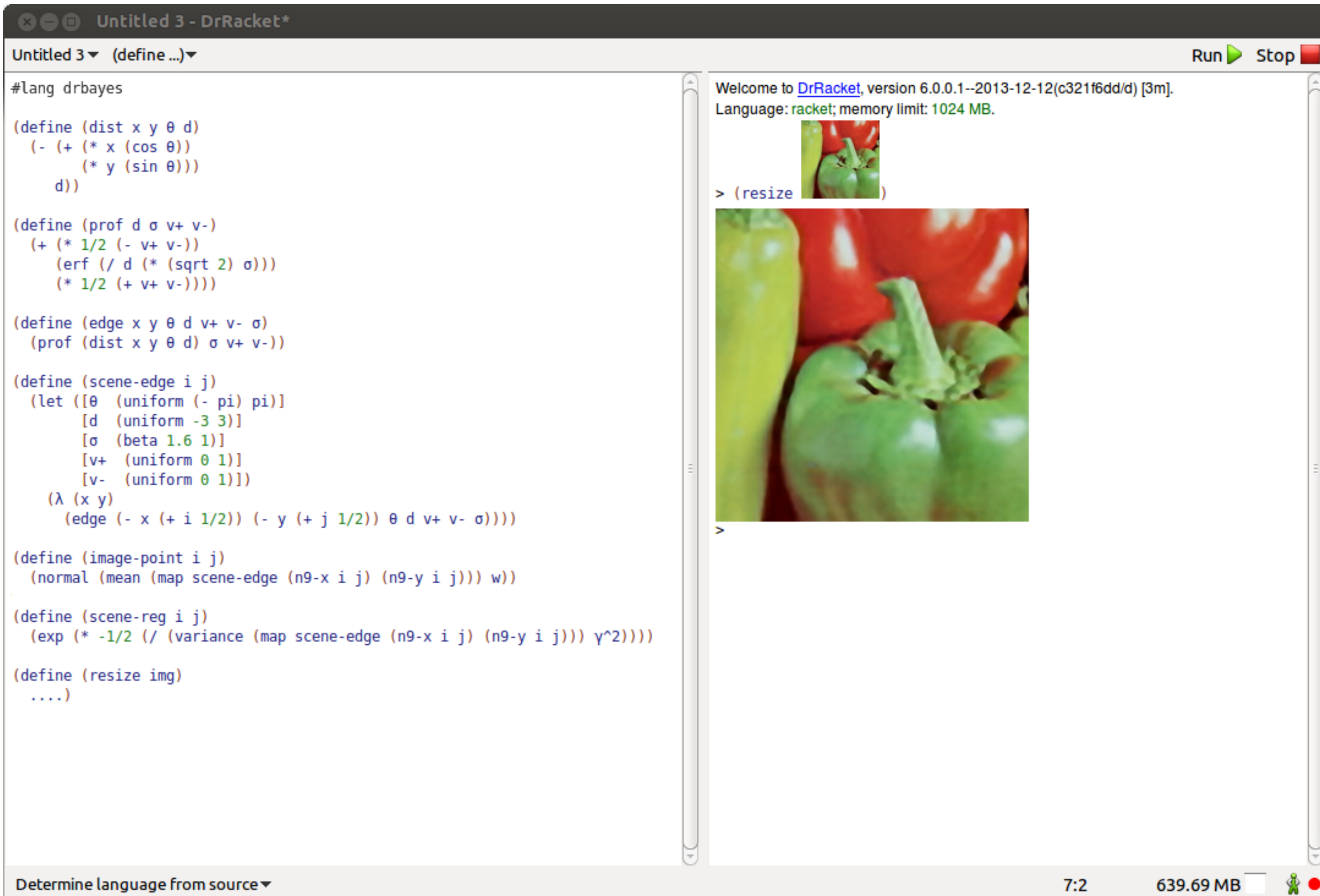
What I Did



Chapter 1
Thesis
Chapter 2
Background
Chapter 3
Related Work
Chapter 4
λ_{ZFC}
Chapter 6
Countable Models and Implementation
Chapter 5
Using λ_{ZFC}
Chapter 8
Preimage Computation Theory
Chapter 9
Preimage Computation Implementation
Chapter 10
Example Programs
Chapter 7
Interlude
Appendix A
Measurability Theorems
Appendix B
Sampling Theorems

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