Some questions about vagueness and metalinguistic uncertainty

Julian Grove

FACTS.lab, November 28, 2022

FACTS.lab, University of Rochester

Vagueness versus metalinguistic uncertainty

Vagueness

(1) The coffee in Rome is expensive. (Kennedy 2007)

Vagueness

(1) The coffee in Rome is expensive. (Kennedy 2007)

$[\![(1)]\!] = \text{cost}(\text{coffeeInRome}) \geq d$ (where d is “vague”)

Vagueness

(1) The coffee in Rome is expensive. (Kennedy 2007)

$[\![(1)]\!] = \mathrm{cost}(\mathrm{coffeeInRome}) \geq d$ (where d is "vague")

Vague predicates, such as *expensive*

- admit borderline cases

(1) The coffee in Rome is expensive. (Kennedy 2007)

$[\![(1)]\!] = \text{cost}(\text{coffeeInRome}) \geq d$ (where d is "vague")

Vague predicates, such as *expensive*

- admit borderline cases
  - Mud Blend: \$1.50/lb ✓

Vagueness

(1) The coffee in Rome is expensive. (Kennedy 2007)

$[\![(1)]\!] = \mathrm{cost}(\mathrm{coffeeInRome}) \geq d$ (where d is "vague")

Vague predicates, such as *expensive*

- admit borderline cases
    - Mud Blend: $1.50/lb ✓
    - Organic Kona: $20/lb ✗

Vagueness

(1) The coffee in Rome is expensive. (Kennedy 2007)

$[\![(1)]\!] = \text{cost}(\text{coffeeInRome}) \geq d$ (where d is "vague")

Vague predicates, such as *expensive*

- admit borderline cases
  - Mud Blend: \$1.50/lb ✓
  - Organic Kona: \$20/lb ✗
  - Swell Start Blend: \$9.25/lb ??

Vagueness

(1) The coffee in Rome is expensive. (Kennedy 2007)

$[\![(1)]\!] = \text{cost}(\text{coffeeInRome}) \geq d$ (where d is "vague")

Vague predicates, such as *expensive*

- admit borderline cases
  - Mud Blend: $1.50/lb ✓
  - Organic Kona: $20/lb ✗
  - Swell Start Blend: $9.25/lb ??
- produce sorites paradoxes:

Vagueness

(1) The coffee in Rome is expensive. (Kennedy 2007)

$[\![(1)]\!] = \text{cost}(\text{coffeeInRome}) \geq d$ (where d is “vague”)

Vague predicates, such as *expensive*

- admit borderline cases
    - Mud Blend: \$1.50/lb ✓
    - Organic Kona: \$20/lb ✗
    - Swell Start Blend: \$9.25/lb ??
- produce sorites paradoxes:

Vagueness

(1) The coffee in Rome is expensive. (Kennedy 2007)

$[\![(1)]\!] = \text{cost}(\text{coffeeInRome}) \geq d$ (where d is “vague”)

Vague predicates, such as *expensive*

- admit borderline cases
  - Mud Blend: $1.50/lb ✓
  - Organic Kona: $20/lb ✗
  - Swell Start Blend: $9.25/lb ??
- produce sorites paradoxes:

P1. A $5 cup of coffee is expensive.

Vagueness

(1) The coffee in Rome is expensive. (Kennedy 2007)

$[\![(1)]\!] = \text{cost}(\text{coffeeInRome}) \geq d$ (where d is "vague")

Vague predicates, such as *expensive*

- admit borderline cases
  - Mud Blend: \$1.50/lb ✓
  - Organic Kona: \$20/lb ✗
  - Swell Start Blend: \$9.25/lb ??
- produce sorites paradoxes:

P1. A \$5 cup of coffee is expensive.

P2. If an expensive cup of coffee were 1 cent cheaper, it would still be expensive.

Vagueness

(1) The coffee in Rome is expensive. (Kennedy 2007)

$[\![(1)]\!] = \mathrm{cost}(\mathrm{coffeeInRome}) \geq d$ (where d is "vague")

Vague predicates, such as *expensive*

- admit borderline cases
  - Mud Blend: \$1.50/lb ✓
  - Organic Kona: \$20/lb ✗
  - Swell Start Blend: \$9.25/lb ??
- produce sorites paradoxes:

P1. A \$5 cup of coffee is expensive.

P2. If an expensive cup of coffee were 1 cent cheaper, it would still be expensive.

C. Therefore, a free cup of coffee is expensive.

Metalinguistic uncertainty

(2) The road is a metric mile long.

(metric mile… somewhere between a kilometer and a (statute) mile)

Metalinguistic uncertainty

(2) The road is a metric mile long.

(metric mile… somewhere between a kilometer and a (statute) mile)

$[\![(2)]\!] = \text{length}(\text{road}) \geq d$ (where d is "uncertain")

Metalinguistic uncertainty

(2) The road is a metric mile long.

(metric mile… somewhere between a kilometer and a (statute) mile)

$[\![(2)]\!] = \text{length}(\text{road}) \geq d$ (where d is "uncertain")

Predicates producing metalinguistic uncertainty, such as *metric mile*

- do not admit borderline cases as easily…

Metalinguistic uncertainty

(2) The road is a metric mile long.

(metric mile… somewhere between a kilometer and a (statute) mile)

$[\![(2)]\!] = \text{length}(\text{road}) \geq d$ (where d is "uncertain")

Predicates producing metalinguistic uncertainty, such as *metric mile*

- do not admit borderline cases as easily…
- do not produce sorites paradoxes:

Metalinguistic uncertainty

(2) The road is a metric mile long.

(metric mile… somewhere between a kilometer and a (statute) mile)

$[\![(2)]\!] = \text{length}(\text{road}) \geq d$ (where d is “uncertain”)

Predicates producing metalinguistic uncertainty, such as *metric mile*

- do not admit borderline cases as easily…
- do not produce sorites paradoxes:

Metalinguistic uncertainty

(2) The road is a metric mile long.

(metric mile… somewhere between a kilometer and a (statute) mile)

$[\![(2)]\!] = \text{length}(\text{road}) \geq d$ (where d is "uncertain")

Predicates producing metalinguistic uncertainty, such as *metric mile*

- do not admit borderline cases as easily…
- do not produce sorites paradoxes:

P1. A 1-mile road is at least a metric mile long.

Metalinguistic uncertainty

(2) The road is a metric mile long.

(metric mile… somewhere between a kilometer and a (statute) mile)

$[\![(2)]\!] = \text{length}(\text{road}) \geq d$ (where d is "uncertain")

Predicates producing metalinguistic uncertainty, such as *metric mile*

- do not admit borderline cases as easily…
- do not produce sorites paradoxes:

P1. A 1-mile road is at least a metric mile long.

P2. If a road at least 1 metric mile long were 1 mm shorter, it would still be at least a metric mile long. ✗

Sorites-like imprecision for uncertainty

However, Lassiter (2011) argues that uncertain factual knowledge can display sorites-like behavior:

> *'There is no real number r such that my belief state allows for the possibility that Big Ben and the Eiffel Tower are r kilometers apart, but excludes the possibility that they are r ± ϵ kilometers apart for sufficiently small ϵ.'*

Sorites-like imprecision for uncertainty

However, Lassiter (2011) argues that uncertain factual knowledge can display sorites-like behavior:

> *'There is no real number r such that my belief state allows for the possibility that Big Ben and the Eiffel Tower are r kilometers apart, but excludes the possibility that they are $r \pm \epsilon$ kilometers apart for sufficiently small ϵ.'*

Still not accessible to sorites arguments…

P2. If the Big Ben and Eiffel Tower are *r* km apart, then they are also 1 mm less then *r* km apart. ✗

Vague parameters are resistant to being made precise

(5) # A \$4.00 cup of coffee is expensive, but a \$3.99 cup of coffee is not expensive.

Vague parameters are resistant to being made precise

(5) # A $4.00 cup of coffee is expensive, but a $3.99 cup of coffee is not expensive.

In contrast, uncertain knowledge can be made certain:

(6) A .93-mile road is 1 metric mile, but a .92-mile road is not 1 metric mile.

Vague parameters support entailments

(3) P1. The coffee in Rome is expensive.

Vague parameters support entailments

(3) P1. The coffee in Rome is expensive.
P2. The coffee in Gothenburg is more expensive than the coffee in Rome.

Vague parameters support entailments

(3) P1. The coffee in Rome is expensive.
P2. The coffee in Gothenburg is more expensive than the coffee in Rome.
C. The coffee in Gothenburg is expensive.

Vague parameters support entailments

(3) P1. The coffee in Rome is expensive.
P2. The coffee in Gothenburg is more expensive than the coffee in Rome.
C. The coffee in Gothenburg is expensive.

This is common to vagueness and metalinguistic uncertainty:

Vague parameters support entailments

(3) P1. The coffee in Rome is expensive.
P2. The coffee in Gothenburg is more expensive than the coffee in Rome.
C. The coffee in Gothenburg is expensive.

This is common to vagueness and metalinguistic uncertainty:

(4) P1. Kenrick Road is at least 1 metric mile long.
P2. East Henrietta is longer than Kenrick.
C. East Henrietta is at least 1 metric mile long.

Vague parameters support entailments

(3) P1. The coffee in Rome is expensive.
P2. The coffee in Gothenburg is more expensive than the coffee in Rome.
C. The coffee in Gothenburg is expensive.

This is common to vagueness and metalinguistic uncertainty:

(4) P1. Kenrick Road is at least 1 metric mile long.
P2. East Henrietta is longer than Kenrick.
C. East Henrietta is at least 1 metric mile long.

In both cases *d* is held constant for the purpose of supporting the entailment from P1 and P2 to C.

Summary

	Vagueness	Metalinguistic uncertainty
Sorites	✓	✗
Resistance to precisification	✓	✗
Support entailments	✓	✓

Summary

	Vagueness	Metalinguistic uncertainty
Sorites	✓	✗
Resistance to precisification	✓	✗
Support entailments	✓	✓

The first two rows suggest that vague parameters are under the control of the surrounding discourse. Surrounding linguistic material modifies them without trouble:

Summary

	Vagueness	Metalinguistic uncertainty
Sorites	✓	✗
Resistance to precisification	✓	✗
Support entailments	✓	✓

The first two rows suggest that vague parameters are under the control of the surrounding discourse. Surrounding linguistic material modifies them without trouble:

- *if* … *then* for sorites

Summary

	Vagueness	Metalinguistic uncertainty
Sorites	✓	✗
Resistance to precisification	✓	✗
Support entailments	✓	✓

The first two rows suggest that vague parameters are under the control of the surrounding discourse. Surrounding linguistic material modifies them without trouble:

- *if* … *then* for sorites
- clause boundaries (perhaps) in the attempted precisification examples

Summary

	Vagueness	Metalinguistic uncertainty
Sorites	✓	✗
Resistance to precisification	✓	✗
Support entailments	✓	✓

The first two rows suggest that vague parameters are under the control of the surrounding discourse. Surrounding linguistic material modifies them without trouble:

- *if* … *then* for sorites
- clause boundaries (perhaps) in the attempted precisification examples

Summary

	Vagueness	Metalinguistic uncertainty
Sorites	✓	✗
Resistance to precisification	✓	✗
Support entailments	✓	✓

The first two rows suggest that vague parameters are under the control of the surrounding discourse. Surrounding linguistic material modifies them without trouble:

- *if* … *then* for sorites
- clause boundaries (perhaps) in the attempted precisification examples

Row 3 suggests that they can be held fixed in certain cases.

Plan

The plan: characterize both vagueness and metalinguistic uncertainty as outcomes of semantic knowledge being probabilistic in nature (Lassiter 2011; Lassiter and Goodman 2013, 2017, i.a.).

Plan

The plan: characterize both vagueness and metalinguistic uncertainty as outcomes of semantic knowledge being probabilistic in nature (Lassiter 2011; Lassiter and Goodman 2013, 2017, i.a.).

- in a pure logical setting, where probabilistic semantic knowledge gives rise to an *applicative functor*

Plan

The plan: characterize both vagueness and metalinguistic uncertainty as outcomes of semantic knowledge being probabilistic in nature (Lassiter 2011; Lassiter and Goodman 2013, 2017, i.a.).

- in a pure logical setting, where probabilistic semantic knowledge gives rise to an *applicative functor*
- and by relying on the composition of applicative functors in order to get a handle on the semantic separation between vagueness and uncertainty

Probabilistic semantics via probabilistic programs

Definition of a probabilistic program

For any type α, a function of type $(\alpha \to r) \to r$ *returns* values of type α.

Definition of a probabilistic program

For any type α, a function of type $(\alpha \to r) \to r$ *returns* values of type α.

- Consumes a *projection function*: some f of type $\alpha \to r$.

Definition of a probabilistic program

For any type α, a function of type $(\alpha \to r) \to r$ *returns* values of type α.

- Consumes a *projection function*: some f of type $\alpha \to r$.
- Results in an r, by summing f over the possible values x of type α, weighting each $f(x)$ by the probability of x.

Definition of a probabilistic program

For any type α, a function of type $(\alpha \to r) \to r$ *returns* values of type α.

- Consumes a *projection function*: some f of type $\alpha \to r$.
- Results in an r, by summing f over the possible values x of type α, weighting each $f(x)$ by the probability of x.

Definition of a probabilistic program

For any type α, a function of type $(\alpha \to r) \to r$ *returns* values of type α.

- Consumes a *projection function*: some f of type $\alpha \to r$.
- Results in an r, by summing f over the possible values x of type α, weighting each $f(x)$ by the probability of x.

Example: $\mathcal{N}(\mu, \sigma) : (r \to r) \to r$

Definition of a probabilistic program

For any type α, a function of type $(\alpha \to r) \to r$ *returns* values of type α.

- Consumes a *projection function*: some f of type $\alpha \to r$.
- Results in an r, by summing f over the possible values x of type α, weighting each $f(x)$ by the probability of x.

Example: $\mathcal{N}(\mu, \sigma) : (r \to r) \to r$

- Represents a normal distribution with mean μ and standard deviation σ.

Definition of a probabilistic program

For any type α, a function of type $(\alpha \to r) \to r$ *returns* values of type α.

- Consumes a *projection function*: some f of type $\alpha \to r$.
- Results in an r, by summing f over the possible values x of type α, weighting each $f(x)$ by the probability of x.

Example: $\mathcal{N}(\mu, \sigma) : (r \to r) \to r$

- Represents a normal distribution with mean μ and standard deviation σ.
- $\mathcal{N}(\mu, \sigma)(f) = \int_{-\infty}^{\infty} \mathrm{PDF}_{\mathcal{N}(\mu,\sigma)}(x) * f(x) dx \qquad (f : r \to r)$

Definition of a probabilistic program

For any type α, a function of type $(\alpha \to r) \to r$ *returns* values of type α.

- Consumes a *projection function*: some f of type $\alpha \to r$.
- Results in an r, by summing f over the possible values x of type α, weighting each $f(x)$ by the probability of x.

Example: $\mathcal{N}(\mu, \sigma) : (r \to r) \to r$

- Represents a normal distribution with mean μ and standard deviation σ.
- $\mathcal{N}(\mu, \sigma)(f) = \int_{-\infty}^{\infty} \mathrm{PDF}_{\mathcal{N}(\mu,\sigma)}(x) * f(x)dx \qquad (f : r \to r)$
  - Result: the weighted average (i.e., *expected value*) of $f(x)$ across the normally distributed values x.

Assembling and using probabilistic programs

We would like to be able to build probabilistic programs representing (vague/uncertain) meanings. We can do this using two ingredients:

Assembling and using probabilistic programs

We would like to be able to build probabilistic programs representing (vague/uncertain) meanings. We can do this using two ingredients:

- a method of turning ordinary logical meanings into probabilistic programs

Assembling and using probabilistic programs

We would like to be able to build probabilistic programs representing (vague/uncertain) meanings. We can do this using two ingredients:

- a method of turning ordinary logical meanings into probabilistic programs
- a method of composing probabilistic programs together, similar to how we compose ordinary natural language meanings by functional application

Probabilistic programs provide an applicative functor

It is convenient to abbreviate $(\alpha \to r) \to r$ as ‘ $\mathsf{P}(\alpha)$ ’, and to then view P itself as a kind of map from types to types.

Probabilistic programs provide an applicative functor

It is convenient to abbreviate $(\alpha \to r) \to r$ as ‘ $\mathsf{P}(\alpha)$ ’, and to then view P itself as a kind of map from types to types.

We can think of the type $\mathsf{P}(\alpha)$ of a probabilistic program as having two parts:

Probabilistic programs provide an applicative functor

It is convenient to abbreviate $(\alpha \to r) \to r$ as ‘ $\mathsf{P}(\alpha)$ ’, and to then view P itself as a kind of map from types to types.

We can think of the type $\mathsf{P}(\alpha)$ of a probabilistic program as having two parts:

1. a part that does *probabilistic* stuff, provided by the P

Probabilistic programs provide an applicative functor

It is convenient to abbreviate $(\alpha \to r) \to r$ as ‘ $\mathsf{P}(\alpha)$ ’, and to then view P itself as a kind of map from types to types.

We can think of the type $\mathsf{P}(\alpha)$ of a probabilistic program as having two parts:

1. a part that does *probabilistic* stuff, provided by the P
2. a part that does regular logical (or *pure*) stuff corresponding to what the program *returns*, provided by the α

Probabilistic programs provide an applicative functor

It is convenient to abbreviate $(\alpha \to r) \to r$ as ‘ $\mathsf{P}(\alpha)$ ’, and to then view P itself as a kind of map from types to types.

We can think of the type $\mathsf{P}(\alpha)$ of a probabilistic program as having two parts:

1. a part that does *probabilistic* stuff, provided by the P
2. a part that does regular logical (or *pure*) stuff corresponding to what the program *returns*, provided by the α

Probabilistic programs provide an applicative functor

It is convenient to abbreviate $(\alpha \to r) \to r$ as ‘ $\mathsf{P}(\alpha)$ ’, and to then view P itself as a kind of map from types to types.

We can think of the type $\mathsf{P}(\alpha)$ of a probabilistic program as having two parts:

1. a part that does *probabilistic* stuff, provided by the P
2. a part that does regular logical (or *pure*) stuff corresponding to what the program *returns*, provided by the α

Viewed this way, the map P is what is known as an *applicative functor*. This means two things…

Applicative functors provide pure programs

First, it allows you to turn any value (of some type α) into a trivial probabilistic program that returns just that value:

Applicative functors provide pure programs

First, it allows you to turn any value (of some type α) into a trivial probabilistic program that returns just that value:

$$\eta : \alpha \rightarrow \mathsf{P}(\alpha) \qquad \text{(`pure')}$$

$$\eta(a) = \lambda f.f(a)$$

Applicative functors provide pure programs

First, it allows you to turn any value (of some type α) into a trivial probabilistic program that returns just that value:

$$\eta : \alpha \to \mathsf{P}(\alpha) \qquad \text{(`pure')}$$

$$\eta(a) = \lambda f.f(a)$$

Example:

$$\text{coffeeInRome} : e \qquad \text{(`the coffee in Rome')}$$

Applicative functors provide pure programs

First, it allows you to turn any value (of some type α) into a trivial probabilistic program that returns just that value:

$$\eta : \alpha \to \mathsf{P}(\alpha) \qquad \text{('pure')}$$

$$\eta(a) = \lambda f.f(a)$$

Example:

$$\text{coffeeInRome} : e \qquad \text{('the coffee in Rome')}$$

$$\eta(\text{coffeeInRome}) : \mathsf{P}(e)$$

Applicative functors provide pure programs

First, it allows you to turn any value (of some type α) into a trivial probabilistic program that returns just that value:

$$\eta : \alpha \rightarrow \mathsf{P}(\alpha) \qquad \text{('pure')}$$
$$\eta(a) = \lambda f.f(a)$$

Example:

$$\begin{array}{rlll} & \text{coffeeInRome} & : e & \text{('the coffee in Rome')} \\ & \eta(\text{coffeeInRome}) & : \mathsf{P}(e) & \\ = & \lambda f.f(\text{coffeeInRome}) & : (e \rightarrow r) \rightarrow r & \end{array}$$

Applicative functors provide pure programs

First, it allows you to turn any value (of some type α) into a trivial probabilistic program that returns just that value:

$$\eta : \alpha \to \mathsf{P}(\alpha) \qquad \text{('pure')}$$
$$\eta(a) = \lambda f.f(a)$$

Example:

$$\begin{aligned} & \text{coffeeInRome} && : e && \text{('the coffee in Rome')} \\ & \eta(\text{coffeeInRome}) && : \mathsf{P}(e) \\ = \; & \lambda f.f(\text{coffeeInRome}) && : (e \to r) \to r \end{aligned}$$

Applicative functors provide pure programs

First, it allows you to turn any value (of some type α) into a trivial probabilistic program that returns just that value:

$$\eta : \alpha \to \mathsf{P}(\alpha) \qquad\qquad (\text{‘pure’})$$

$$\eta(a) = \lambda f.f(a)$$

Example:

$$\begin{aligned}
&\text{coffeeInRome} && : e && (\text{‘the coffee in Rome’})\\
&\eta(\text{coffeeInRome}) && : \mathsf{P}(e) &&\\
= \ &\lambda f.f(\text{coffeeInRome}) && : (e \to r) \to r &&
\end{aligned}$$

(The probabilistic program that returns the coffee in Rome with a probability of 1.)

Applicative functors allow you to compose programs together

Given programs

- $m : \mathsf{P}(\alpha \to \beta)$
- $n : \mathsf{P}(\alpha)$

you can compose m and n by feeding the values returned by n to the functions returned by m.

Applicative functors allow you to compose programs together

Given programs

- $m : \mathsf{P}(\alpha \to \beta)$
- $n : \mathsf{P}(\alpha)$

you can compose m and n by feeding the values returned by n to the functions returned by m.

$$(\circledast) : \mathsf{P}(\alpha \to \beta) \to \mathsf{P}(\alpha) \to \mathsf{P}(\beta) \quad \text{(‘sequential application’)}$$
$$m \circledast n = \lambda f.m(\lambda x.n(\lambda y.f(x(y))))$$

Applicative functors allow you to compose programs together

Given programs

- $m : \mathsf{P}(\alpha \to \beta)$
- $n : \mathsf{P}(\alpha)$

you can compose *m* and *n* by feeding the values returned by *n* to the functions returned by *m*.

$$(\circledast) : \mathsf{P}(\alpha \to \beta) \to \mathsf{P}(\alpha) \to \mathsf{P}(\beta) \quad \text{(‘sequential application’)}$$

$$m \circledast n = \lambda f.m(\lambda x.n(\lambda y.f(x(y))))$$

“Run *m* to compute *x*. Then run *n* to compute *y*. Then apply *x* to *y*.”

Computing probabilities

Given a probabilistic program m of type $\mathsf{P}(t)$ (i.e., returning truth values), we may use it to compute a probability:

Computing probabilities

Given a probabilistic program m of type $\mathsf{P}(t)$ (i.e., returning truth values), we may use it to compute a probability:

$$Pr : \mathsf{P}(t) \rightarrow r$$

Computing probabilities

Given a probabilistic program m of type $\mathsf{P}(t)$ (i.e., returning truth values), we may use it to compute a probability:

$$Pr : \mathsf{P}(t) \to r$$
$$Pr(m) = \frac{m(\mathbb{1})}{m(\lambda b.1)}$$

Computing probabilities

Given a probabilistic program m of type $\mathsf{P}(t)$ (i.e., returning truth values), we may use it to compute a probability:

$$Pr : \mathsf{P}(t) \rightarrow r$$
$$Pr(m) = \frac{m(\mathbb{1})}{m(\lambda b.1)}$$

Computing probabilities

Given a probabilistic program m of type $\mathsf{P}(t)$ (i.e., returning truth values), we may use it to compute a probability:

$$Pr : \mathsf{P}(t) \to r$$
$$Pr(m) = \frac{m(\mathbb{1})}{m(\lambda b.1)}$$

- $\mathbb{1} : t \to r$ is an indicator function:

Computing probabilities

Given a probabilistic program m of type $\mathsf{P}(t)$ (i.e., returning truth values), we may use it to compute a probability:

$$Pr : \mathsf{P}(t) \to r$$

$$Pr(m) = \frac{m(\mathbb{1})}{m(\lambda b.1)}$$

- $\mathbb{1} : t \to r$ is an indicator function:
  - $\mathbb{1}(\top) = 1$

Computing probabilities

Given a probabilistic program m of type $\mathsf{P}(t)$ (i.e., returning truth values), we may use it to compute a probability:

$$Pr : \mathsf{P}(t) \rightarrow r$$
$$Pr(m) = \frac{m(\mathbb{1})}{m(\lambda b.1)}$$

- $\mathbb{1} : t \rightarrow r$ is an indicator function:
  - $\mathbb{1}(\top) = 1$
  - $\mathbb{1}(\bot) = 0$

Computing probabilities

Given a probabilistic program m of type $\mathsf{P}(t)$ (i.e., returning truth values), we may use it to compute a probability:

$$Pr : \mathsf{P}(t) \rightarrow r$$
$$Pr(m) = \frac{m(\mathbb{1})}{m(\lambda b.1)}$$

- $\mathbb{1} : t \rightarrow r$ is an indicator function:
    - $\mathbb{1}(\top) = 1$
    - $\mathbb{1}(\bot) = 0$
- In the above, it picks out the mass assigned to $\top$.

Computing probabilities

Given a probabilistic program m of type $\mathsf{P}(t)$ (i.e., returning truth values), we may use it to compute a probability:

$$Pr : \mathsf{P}(t) \rightarrow r$$
$$Pr(m) = \frac{m(\mathbb{1})}{m(\lambda b.1)}$$

- $\mathbb{1} : t \rightarrow r$ is an indicator function:
  - $\mathbb{1}(\top) = 1$
  - $\mathbb{1}(\bot) = 0$
- In the above, it picks out the mass assigned to $\top$.
- $m(\lambda b.1)$ is the *measure* of m: it is m's *total mass*.

Computing probabilities

Given a probabilistic program m of type $\mathsf{P}(t)$ (i.e., returning truth values), we may use it to compute a probability:

$$Pr : \mathsf{P}(t) \to r$$
$$Pr(m) = \frac{m(\mathbb{1})}{m(\lambda b.1)}$$

- $\mathbb{1} : t \to r$ is an indicator function:
  - $\mathbb{1}(\top) = 1$
  - $\mathbb{1}(\bot) = 0$
- In the above, it picks out the mass assigned to $\top$.
- $m(\lambda b.1)$ is the *measure* of m: it is m's *total mass.*
- So, $Pr(m)$ is the probability that m returns $\top$.

Probabilistic semantics for vagueness

An example

(1) The coffee in Rome is expensive.

An example

(1) The coffee in Rome is expensive.

$[\![\text{the coffee in Rome}]\!] : \mathsf{P}(e)$
$[\![\text{the coffee in Rome}]\!] = \eta(\text{coffeeInRome})$

An example

(1) The coffee in Rome is expensive.

$[\![\text{the coffee in Rome}]\!] : \mathsf{P}(e)$
$[\![\text{the coffee in Rome}]\!] = \eta(\text{coffeeInRome})$

$[\![\text{expensive}]\!] : \mathsf{P}(e \rightarrow t)$
$[\![\text{expensive}]\!] = \lambda f.\mathcal{N}(\mu_{exp}, \sigma_{exp})(\lambda d.f(\lambda x.\text{cost}(x) \geq d))$

An example

(1) The coffee in Rome is expensive.

$[\![\text{the coffee in Rome}]\!] : \mathsf{P}(e)$

$[\![\text{the coffee in Rome}]\!] = \eta(\text{coffeeInRome})$

$[\![\text{expensive}]\!] : \mathsf{P}(e \longrightarrow t)$

$[\![\text{expensive}]\!] = \lambda f.\mathcal{N}(\mu_{exp}, \sigma_{exp})(\lambda d.f(\lambda x.\text{cost}(x) \geq d))$

$[\![(1)]\!] : \mathsf{P}(t)$

$$[\![(1)]\!] = [\![\text{expensive}]\!] \circledast [\![\text{the coffee in Rome}]\!]$$
$$= \lambda f.\mathcal{N}(\mu_{exp}, \sigma_{exp})(\lambda d.f(\text{cost}(\text{coffeeInRome}) \geq d))$$

An example

(1) The coffee in Rome is expensive.

$[\![\text{the coffee in Rome}]\!] : \mathsf{P}(e)$
$[\![\text{the coffee in Rome}]\!] = \eta(\text{coffeeInRome})$

$[\![\text{expensive}]\!] : \mathsf{P}(e \longrightarrow t)$
$[\![\text{expensive}]\!] = \lambda f.\mathcal{N}(\mu_{exp}, \sigma_{exp})(\lambda d.f(\lambda x.\text{cost}(x) \geq d))$

$[\![(1)]\!] : \mathsf{P}(t)$
$[\![(1)]\!] = [\![\text{expensive}]\!] \circledast [\![\text{the coffee in Rome}]\!]$
$= \lambda f.\mathcal{N}(\mu_{exp}, \sigma_{exp})(\lambda d.f(\text{cost}(\text{coffeeInRome}) \geq d))$

If, for example, cost(coffeeInRome) $= \mu_{exp}$, then $Pr([\![(1)]\!]) = 0.5$.

Sorites

(5) If an expensive cup of coffee were 1 cent cheaper, it would still be expensive.

(5) If an expensive cup of coffee were 1 cent cheaper, it would still be expensive.

We need a meaning for *if*‼

Factoring by a weight / observing a premise

$$factor : r \longrightarrow \mathsf{P}(\diamond)$$

Factoring by a weight / observing a premise

$$\begin{aligned} \mathit{factor} &: r \longrightarrow \mathsf{P}(\diamond) \\ \mathit{factor}(x)(f) &= x * f(\diamond) \end{aligned}$$

Factoring by a weight / observing a premise

$$
\begin{aligned}
\mathit{factor} &: r \longrightarrow \mathsf{P}(\diamond) \\
\mathit{factor}(x)(f) &= x * f(\diamond)
\end{aligned}
$$

$$
\mathit{observe} : r \longrightarrow \mathsf{P}(\diamond)
$$

Factoring by a weight / observing a premise

$$\begin{aligned} \mathit{factor} &: r \longrightarrow \mathsf{P}(\diamond) \\ \mathit{factor}(x)(f) &= x * f(\diamond) \end{aligned}$$

$$\begin{aligned} \mathit{observe} &: r \longrightarrow \mathsf{P}(\diamond) \\ \mathit{observe}(\phi)(f) &= \mathit{factor}(\mathbb{1}(\phi))(f) \end{aligned}$$

Factoring by a weight / observing a premise

$$\begin{aligned} \mathit{factor} &: r \longrightarrow \mathsf{P}(\diamond) \\ \mathit{factor}(x)(f) &= x * f(\diamond) \end{aligned}$$

$$\begin{aligned} \mathit{observe} &: r \longrightarrow \mathsf{P}(\diamond) \\ \mathit{observe}(\phi)(f) &= \mathit{factor}(\mathbb{1}(\phi))(f) \\ &= \mathbb{1}(\phi) * f(\diamond) \end{aligned}$$

Sorites (cont'd 1)

(5) If an expensive cup of coffee were 1 cent cheaper, it would still be expensive.

Sorites (cont'd 1)

(5) If an expensive cup of coffee were 1 cent cheaper, it would still be expensive.

Sorites (cont'd 1)

(5) If an expensive cup of coffee were 1 cent cheaper, it would still be expensive.

$[\![\mathit{if}]\!] : (w \to t) \to (w \to t) \to \mathsf{P}(w) \to t$

w is the type of possible worlds (of some kind, e.g., degrees of cost)

Sorites (cont'd 1)

(5) If an expensive cup of coffee were 1 cent cheaper, it would still be expensive.

$[\![if]\!] : (w \to t) \to (w \to t) \to \mathsf{P}(w) \to t$

w is the type of possible worlds (of some kind, e.g., degrees of cost)

$[\![if]\!](\phi)(\psi)(mb) =$

$$Pr(\lambda f.mb(\lambda w.observe(\phi(w))(\lambda \diamond.f(\psi(w))))) \geq r_{certainty}$$

(5) If an expensive cup of coffee were 1 cent cheaper, it would still be expensive.

$[\![if]\!] : (w \to t) \to (w \to t) \to \mathsf{P}(w) \to t$

w is the type of possible worlds (of some kind, e.g., degrees of cost)

$[\![if]\!](\phi)(\psi)(mb) =$

$$Pr(\lambda f.mb(\lambda w.observe(\phi(w))(\lambda\diamond.f(\psi(w))))) \geq r_{certainty}$$

"Given some distribution over worlds *mb*, the probability that ψ is true after filtering out the worlds where ϕ is false is greater than some required threshold of certainty $r_{certainty}$."

Sorites (cont'd 2)

(5) If an expensive cup of coffee were 1 cent cheaper, it would still be expensive.

Sorites (cont'd 2)

(5) If an expensive cup of coffee were 1 cent cheaper, it would still be expensive.

Let's fix w to $r \times r$ — the type of pairs of degrees representing costs.

- r on the left: represents the cost of different cups of coffee
- r on the right: represents the threshold for *expensive*

Sorites (cont'd 2)

(5) If an expensive cup of coffee were 1 cent cheaper, it would still be expensive.

Let's fix w to $r \times r$ — the type of pairs of degrees representing costs.

- r on the left: represents the cost of different cups of coffee
- r on the right: represents the threshold for *expensive*

$[\![\textit{an expensive cup were 1 cent cheaper}]\!] : r \times r \to t$
$[\![\textit{an expensive cup were 1 cent cheaper}]\!] = \lambda\langle d, d'\rangle . d \geq d' - 0.01$

Sorites (cont'd 2)

(5) If an expensive cup of coffee were 1 cent cheaper, it would still be expensive.

Let's fix w to $r \times r$ — the type of pairs of degrees representing costs.

- r on the left: represents the cost of different cups of coffee
- r on the right: represents the threshold for *expensive*

$$[\![\textit{an expensive cup were 1 cent cheaper}]\!] : r \times r \rightarrow t$$
$$[\![\textit{an expensive cup were 1 cent cheaper}]\!] = \lambda \langle d, d' \rangle . d \geq d' - 0.01$$

$$[\![\textit{it would still be expensive}]\!] : r \times r \rightarrow t$$
$$[\![\textit{it would still be expensive}]\!] = \lambda \langle d, d' \rangle . d \geq d'$$

Sorites (cont'd 3)

(5) If an expensive cup of coffee were 1 cent cheaper, it would still be expensive.

Sorites (cont'd 3)

(5) If an expensive cup of coffee were 1 cent cheaper, it would still be expensive.

$[\![(5)]\!] : t$

$[\![(5)]\!] = Pr(\lambda f.mb(\lambda\langle d, d'\rangle.\mathbb{1}(d \geq d' - 0.01) * f(d \geq d'))) \geq r_{certainty}$

Sorites (cont'd 3)

(5) If an expensive cup of coffee were 1 cent cheaper, it would still be expensive.

$[\![(5)]\!] : t$

$[\![(5)]\!] = Pr(\lambda f.mb(\lambda\langle d, d'\rangle.\mathbb{1}(d \geq d' - 0.01) * f(d \geq d'))) \geq r_{certainty}$

"If you take the mass of *mb* where $d \geq d' - 0.01$, the proportion of this mass where $d \geq d'$, as well, is greater than the certainty threshold."

Sorites (cont'd 3)

(5) If an expensive cup of coffee were 1 cent cheaper, it would still be expensive.

$[\![(5)]\!] : t$

$[\![(5)]\!] = Pr(\lambda f.mb(\lambda\langle d, d'\rangle.\mathbb{1}(d \geq d' - 0.01) * f(d \geq d'))) \geq r_{certainty}$

"If you take the mass of *mb* where $d \geq d' - 0.01$, the proportion of this mass where $d \geq d'$, as well, is greater than the certainty threshold."

For example, if d and d' are independently normally distributed with the same mean, this will always be ≥ 0.5. For $\sigma = 1$, it is > 0.99.

Entailments

(3) P1. The coffee in Rome is expensive.
P2. The coffee in Gothenburg is more expensive than the coffee in Rome.
C. The coffee in Gothenburg is expensive.

Entailments

(3) P1. The coffee in Rome is expensive.
P2. The coffee in Gothenburg is more expensive than the coffee in Rome.
C. The coffee in Gothenburg is expensive.

We need an operation to perform discourse update:

$$update : \mathsf{P}(w) \to (w \to t) \to \mathsf{P}(w)$$
$$update(c)(\phi) = \lambda f.c(\lambda w.observe(\phi(w))(\lambda\diamond.f(w)))$$

Entailments

(3) P1. The coffee in Rome is expensive.
P2. The coffee in Gothenburg is more expensive than the coffee in Rome.
C. The coffee in Gothenburg is expensive.

We need an operation to perform discourse update:

$$update : \mathsf{P}(w) \to (w \to t) \to \mathsf{P}(w)$$

$$update(c)(\phi) = \lambda f.c(\lambda w.observe(\phi(w))(\lambda\diamond.f(w)))$$

"Given a starting discourse c and a proposition ϕ to update it with, $update(c)(\phi)$ is just like c, except that worlds where ϕ is false are assigned a probability of 0."

Entailments (cont'd 1)

(3) P1. The coffee in Rome is expensive.
P2. The coffee in Gothenburg is more expensive than the coffee in Rome.
C. The coffee in Gothenburg is expensive.

Entailments (cont'd 1)

(3) P1. The coffee in Rome is expensive.
P2. The coffee in Gothenburg is more expensive than the coffee in Rome.
C. The coffee in Gothenburg is expensive.

In this case, let's consider w to be $r \times r \times r$.

Entailments (cont'd 1)

(3) P1. The coffee in Rome is expensive.
P2. The coffee in Gothenburg is more expensive than the coffee in Rome.
C. The coffee in Gothenburg is expensive.

In this case, let's consider w to be $r \times r \times r$.

- the first r: represents the cost of coffee in Rome

Entailments (cont'd 1)

(3) P1. The coffee in Rome is expensive.
P2. The coffee in Gothenburg is more expensive than the coffee in Rome.
C. The coffee in Gothenburg is expensive.

In this case, let's consider w to be $r \times r \times r$.

- the first r: represents the cost of coffee in Rome
- the second r: represents the cost of coffee in Gothenburg

Entailments (cont'd 1)

(3) P1. The coffee in Rome is expensive.
P2. The coffee in Gothenburg is more expensive than the coffee in Rome.
C. The coffee in Gothenburg is expensive.

In this case, let's consider w to be $r \times r \times r$.

- the first r: represents the cost of coffee in Rome
- the second r: represents the cost of coffee in Gothenburg
- the third r: represents the threshold for *expensive*

Entailments (cont'd 1)

(3) P1. The coffee in Rome is expensive.
P2. The coffee in Gothenburg is more expensive than the coffee in Rome.
C. The coffee in Gothenburg is expensive.

In this case, let's consider w to be $r \times r \times r$.

- the first r: represents the cost of coffee in Rome
- the second r: represents the cost of coffee in Gothenburg
- the third r: represents the threshold for *expensive*

Entailments (cont'd 1)

(3) P1. The coffee in Rome is expensive.
P2. The coffee in Gothenburg is more expensive than the coffee in Rome.
C. The coffee in Gothenburg is expensive.

In this case, let's consider w to be $r \times r \times r$.

- the first r: represents the cost of coffee in Rome
- the second r: represents the cost of coffee in Gothenburg
- the third r: represents the threshold for *expensive*

$[\![\text{P1}]\!] : r \times r \times r \to t$
$[\![\text{P1}]\!] = \lambda\langle r, g, d\rangle . r \geq d$

Entailments (cont'd 1)

(3) P1. The coffee in Rome is expensive.
P2. The coffee in Gothenburg is more expensive than the coffee in Rome.
C. The coffee in Gothenburg is expensive.

In this case, let's consider w to be $r \times r \times r$.

- the first r: represents the cost of coffee in Rome
- the second r: represents the cost of coffee in Gothenburg
- the third r: represents the threshold for *expensive*

$[\![\mathrm{P1}]\!] : r \times r \times r \rightarrow t$
$[\![\mathrm{P1}]\!] = \lambda\langle r, g, d\rangle . r \geq d$

$[\![\mathrm{P2}]\!] : r \times r \times r \rightarrow t$
$[\![\mathrm{P2}]\!] = \lambda\langle r, g, d\rangle . g > r$

Entailments (cont'd 2)

(3) P1. The coffee in Rome is expensive.
P2. The coffee in Gothenburg is more expensive than the coffee in Rome.
C. The coffee in Gothenburg is expensive.

Entailments (cont'd 2)

(3) P1. The coffee in Rome is expensive.
P2. The coffee in Gothenburg is more expensive than the coffee in Rome.
C. The coffee in Gothenburg is expensive.

$$update(update(c)(\llbracket \text{P1} \rrbracket))(\llbracket \text{P2} \rrbracket) =$$

$$\lambda f.c(\lambda \langle r, g, d \rangle . \mathbb{1}(r \geq d) * \mathbb{1}(g > r) * f(r, g, d))$$

Entailments (cont'd 2)

(3) P1. The coffee in Rome is expensive.
P2. The coffee in Gothenburg is more expensive than the coffee in Rome.
C. The coffee in Gothenburg is expensive.

$\mathit{update}(\mathit{update}(c)(\llbracket \mathrm{P1} \rrbracket))(\llbracket \mathrm{P2} \rrbracket) =$

$$\lambda f.c(\lambda\langle r, g, d\rangle.\mathbb{1}(r \geq d) * \mathbb{1}(g > r) * f(r, g, d))$$

"The context just like c, except that the only worlds with non-zero probabilities are those where $r \geq d$ and $g > r$ (and, hence, $g > d$)."

Summary

- Vague predicates are susceptible to sorites because of the lexical semantics of certain surrounding linguistic expressions, for example, *if*. Such expressions have higher-order meanings that allow them to control vague parameters.

Summary

- Vague predicates are susceptible to sorites because of the lexical semantics of certain surrounding linguistic expressions, for example, *if*. Such expressions have higher-order meanings that allow them to control vague parameters.
- Vague predicates support entailments that hold their parameters fixed because of the semantics of discourse update. The update operation keeps vague parameters in scope.

Summary

- Vague predicates are susceptible to sorites because of the lexical semantics of certain surrounding linguistic expressions, for example, *if*. Such expressions have higher-order meanings that allow them to control vague parameters.
- Vague predicates support entailments that hold their parameters fixed because of the semantics of discourse update. The update operation keeps vague parameters in scope.

Summary

- Vague predicates are susceptible to sorites because of the lexical semantics of certain surrounding linguistic expressions, for example, *if*. Such expressions have higher-order meanings that allow them to control vague parameters.
- Vague predicates support entailments that hold their parameters fixed because of the semantics of discourse update. The update operation keeps vague parameters in scope.

What makes metalinguistic uncertainty different?

Probabilistic semantics for metalinguistic uncertainty

Applicatives compose

A convenience of using applicative functors is that they compose:

Applicatives compose

A convenience of using applicative functors is that they compose:

If $\mathsf{A}_1(\alpha)$ is an applicative and $\mathsf{A}_2(\alpha)$ is an applicative, then $\mathsf{A}_1(\mathsf{A}_2(\alpha))$ is also an applicative.

Applicatives compose

A convenience of using applicative functors is that they compose:

If $\mathsf{A}_1(\alpha)$ is an applicative and $\mathsf{A}_2(\alpha)$ is an applicative, then $\mathsf{A}_1(\mathsf{A}_2(\alpha))$ is also an applicative.

Useful, because it allows us to think of metalinguistic uncertainty as *higher order*: it is a property that our knowledge of any expression can have, including vague ones.

Applicatives compose

A convenience of using applicative functors is that they compose:

If $\mathsf{A}_1(\alpha)$ is an applicative and $\mathsf{A}_2(\alpha)$ is an applicative, then $\mathsf{A}_1(\mathsf{A}_2(\alpha))$ is also an applicative.

Useful, because it allows us to think of metalinguistic uncertainty as *higher order*: it is a property that our knowledge of any expression can have, including vague ones.

This perspective can be reflected in the type of an expression with metalinguistic uncertainty taken into account: rather than $\mathsf{P}(\alpha)$, it is now of type $\mathsf{P}(\mathsf{P}(\alpha))$.

Applicatives compose

A convenience of using applicative functors is that they compose:

If $\mathsf{A}_1(\alpha)$ is an applicative and $\mathsf{A}_2(\alpha)$ is an applicative, then $\mathsf{A}_1(\mathsf{A}_2(\alpha))$ is also an applicative.

Useful, because it allows us to think of metalinguistic uncertainty as *higher order*: it is a property that our knowledge of any expression can have, including vague ones.

This perspective can be reflected in the type of an expression with metalinguistic uncertainty taken into account: rather than $\mathsf{P}(\alpha)$, it is now of type $\mathsf{P}(\mathsf{P}(\alpha))$.

- Vagueness: $\mathsf{P}(\mathsf{P}(\alpha))$

Applicatives compose

A convenience of using applicative functors is that they compose:

If $\mathsf{A}_1(\alpha)$ is an applicative and $\mathsf{A}_2(\alpha)$ is an applicative, then $\mathsf{A}_1(\mathsf{A}_2(\alpha))$ is also an applicative.

Useful, because it allows us to think of metalinguistic uncertainty as *higher order*: it is a property that our knowledge of any expression can have, including vague ones.

This perspective can be reflected in the type of an expression with metalinguistic uncertainty taken into account: rather than $\mathsf{P}(\alpha)$, it is now of type $\mathsf{P}(\mathsf{P}(\alpha))$.

- Vagueness: $\mathsf{P}(\mathsf{P}(\alpha))$
- Uncertainty: $\mathsf{P}(\mathsf{P}(\alpha))$

An example

(2) The road is a metric mile long.

An example

(2) The road is a metric mile long.

$[\![\text{the road}]\!] : \mathsf{P}(\mathsf{P}(e))$

$[\![\text{the road}]\!] = \eta(\eta(\text{road}))$

An example

(2) The road is a metric mile long.

$[\![\text{the road}]\!] : \mathsf{P}(\mathsf{P}(e))$
$[\![\text{the road}]\!] = \eta(\eta(\text{road}))$

$[\![\text{metric mile long}]\!] : \mathsf{P}(e \to t)$
$[\![\text{metric mile long}]\!] = \lambda f.\mathcal{N}(\mu_{mm}, \sigma_{mm})(\lambda d.f(\eta(\lambda x.\text{length}(x) \geq d)))$

An example

(2) The road is a metric mile long.

$[\![\text{the road}]\!] : \mathsf{P}(\mathsf{P}(e))$

$[\![\text{the road}]\!] = \eta(\eta(\text{road}))$

$[\![\text{metric mile long}]\!] : \mathsf{P}(e \longrightarrow t)$

$[\![\text{metric mile long}]\!] = \lambda f.\mathcal{N}(\mu_{mm}, \sigma_{mm})(\lambda d.f(\eta(\lambda x.\text{length}(x) \geq d)))$

$[\![(2)]\!] : \mathsf{P}(\mathsf{P}(t))$

$$[\![(2)]\!] = [\![\text{metric mile long}]\!] \circledast [\![\text{road}]\!]$$
$$= \lambda f.\mathcal{N}(\mu_{exp}, \sigma_{exp})(\lambda d.f(\eta(\text{length}(\text{road}) \geq d)))$$

Why no sorites?

The type provided for *if*: $(w \rightarrow t) \rightarrow (w \rightarrow t) \rightarrow \mathsf{P}(w) \rightarrow t$

Why no sorites?

The type provided for *if*: $(w \to t) \to (w \to t) \to \mathsf{P}(w) \to t$

Its third argument (the modal base) is not high-order enough.

Why no sorites?

The type provided for *if*: $(w \to t) \to (w \to t) \to \mathsf{P}(w) \to t$

Its third argument (the modal base) is not high-order enough.

Currently encoded as a brute lexical fact.

Why no sorites?

The type provided for *if*: $(w \to t) \to (w \to t) \to \mathsf{P}(w) \to t$

Its third argument (the modal base) is not high-order enough.
Currently encoded as a brute lexical fact.

Hypothetical constraint on lexical meanings:

$\mathsf{P}(\mathsf{P}(\alpha))$ can't occur in a negative position in an expression's type.

Entailments?

We need only adjust the type of the *update*, using η:

$$update : \mathsf{P}(\mathsf{P}(w) \to (w \to t) \to \mathsf{P}(w))$$

$$update = \eta(\lambda c, \phi, f.c(\lambda w.observe(\phi(w))(\lambda \diamond.f(w))))$$

Summary

Under the current picture, the phenomena of vagueness arise from two aspects of semantic knowledge conspiring:

Summary

Under the current picture, the phenomena of vagueness arise from two aspects of semantic knowledge conspiring:

- the semantic types of linguistic expressions like *if*

Summary

Under the current picture, the phenomena of vagueness arise from two aspects of semantic knowledge conspiring:

- the semantic types of linguistic expressions like *if*
- the encoding of vague probabilistic knowledge on an "inner" applicative layer ($\mathsf{P}(\mathsf{P}(\alpha))$)

Summary

Under the current picture, the phenomena of vagueness arise from two aspects of semantic knowledge conspiring:

- the semantic types of linguistic expressions like *if*
- the encoding of vague probabilistic knowledge on an "inner" applicative layer ($\mathsf{P}(\mathsf{P}(\alpha))$)

Summary

Under the current picture, the phenomena of vagueness arise from two aspects of semantic knowledge conspiring:

- the semantic types of linguistic expressions like *if*
- the encoding of vague probabilistic knowledge on an “inner” applicative layer ($\mathsf{P}(\mathsf{P}(\alpha))$)

This makes room for some probabilistic knowledge not participate in these phenomena — encode them on the “outer” layer ($\mathsf{P}(\mathsf{P}(\alpha))$).

References

Kennedy, Christopher. 2007. Vagueness and grammar: the semantics of relative and absolute gradable adjectives. *Linguistics and Philosophy* 30.1, pp. 1–45. DOI: 10.1007/s10988-006-9008-0.

Lassiter, Daniel. 2011. Vagueness as Probabilistic Linguistic Knowledge. *Vagueness in Communication*. Ed. by Rick Nouwen et al. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, pp. 127–150. DOI: 10.1007/978-3-642-18446-8_8.

Lassiter, Daniel and Noah D. Goodman. 2013. Context, scale structure, and statistics in the interpretation of positive-form adjectives. *Semantics and Linguistic Theory* 23.0, pp. 587–610. DOI: 10.3765/salt.v23i0.2658.

Lassiter, Daniel and Noah D. Goodman. 2017. Adjectival vagueness in a Bayesian model of interpretation. *Synthese* 194.10, pp. 3801–3836. DOI: 10.1007/s11229-015-0786-1.