Probabilistic compositional semantics, purely

Julian Grove and Jean-Philippe Bernardy

CLASP Seminar, November 10, 2021

CLASP, University of Gothenburg

Outline

Introduction

Probabilistic semantics

In the last decade, lots of effort to connect formal semantics to mathematically explicit models of pragmatic reasoning…

Probabilistic semantics

In the last decade, lots of effort to connect formal semantics to mathematically explicit models of pragmatic reasoning…

- Rational Speech Act (RSA) models to formalize Gricean pragmatics (Goodman and Stuhlmüller, 2013; Lassiter and Goodman, 2013; Goodman and Frank, 2016; Lassiter and Goodman, 2017)

Probabilistic semantics

In the last decade, lots of effort to connect formal semantics to mathematically explicit models of pragmatic reasoning…

- Rational Speech Act (RSA) models to formalize Gricean pragmatics (Goodman and Stuhlmüller, 2013; Lassiter and Goodman, 2013; Goodman and Frank, 2016; Lassiter and Goodman, 2017)

Probabilistic semantics

In the last decade, lots of effort to connect formal semantics to mathematically explicit models of pragmatic reasoning…

- Rational Speech Act (RSA) models to formalize Gricean pragmatics (Goodman and Stuhlmüller, 2013; Lassiter and Goodman, 2013; Goodman and Frank, 2016; Lassiter and Goodman, 2017)

…generally, by dropping typed λ-calculus and encoding meanings in terms of probabilistic programming languages.

Probabilistic semantics

In the last decade, lots of effort to connect formal semantics to mathematically explicit models of pragmatic reasoning…

- Rational Speech Act (RSA) models to formalize Gricean pragmatics (Goodman and Stuhlmüller, 2013; Lassiter and Goodman, 2013; Goodman and Frank, 2016; Lassiter and Goodman, 2017)

…generally, by dropping typed λ-calculus and encoding meanings in terms of probabilistic programming languages.

- Church (Goodman et al., 2008)

Probabilistic semantics

In the last decade, lots of effort to connect formal semantics to mathematically explicit models of pragmatic reasoning…

- Rational Speech Act (RSA) models to formalize Gricean pragmatics (Goodman and Stuhlmüller, 2013; Lassiter and Goodman, 2013; Goodman and Frank, 2016; Lassiter and Goodman, 2017)

…generally, by dropping typed λ-calculus and encoding meanings in terms of probabilistic programming languages.

- Church (Goodman et al., 2008)

Probabilistic semantics

In the last decade, lots of effort to connect formal semantics to mathematically explicit models of pragmatic reasoning…

- Rational Speech Act (RSA) models to formalize Gricean pragmatics (Goodman and Stuhlmüller, 2013; Lassiter and Goodman, 2013; Goodman and Frank, 2016; Lassiter and Goodman, 2017)

…generally, by dropping typed λ-calculus and encoding meanings in terms of probabilistic programming languages.

- Church (Goodman et al., 2008)

Such programming languages are often *impure*: they allow for probabilistic effects, like sampling and marginalization, to occur at any point in a program.

Today's talk

Goal: show how a probabilistic semantics for natural language can be presented using only the simply typed λ-calculus (with products).

Today's talk

Goal: show how a probabilistic semantics for natural language can be presented using only the simply typed λ-calculus (with products).

- Achieve a seamless integration with approaches to meaning based on higher-order logic.

Today's talk

Goal: show how a probabilistic semantics for natural language can be presented using only the simply typed λ-calculus (with products).

- Achieve a seamless integration with approaches to meaning based on higher-order logic.

Today's talk

Goal: show how a probabilistic semantics for natural language can be presented using only the simply typed λ-calculus (with products).

- Achieve a seamless integration with approaches to meaning based on higher-order logic.

End up with a characterization of meanings as *probabilistic programs*, which are, nevertheless, pure (i.e., no real probabilistic effects).

Today's talk

Goal: show how a probabilistic semantics for natural language can be presented using only the simply typed λ-calculus (with products).

- Achieve a seamless integration with approaches to meaning based on higher-order logic.

End up with a characterization of meanings as *probabilistic programs*, which are, nevertheless, pure (i.e., no real probabilistic effects).

Such programs *describe* probability distributions over logical meanings.

Formal semantics

Logics and sets

Two strategies to formally interpret natural language, inherited from Montague:

Two strategies to formally interpret natural language, inherited from Montague:

- direct: right into set theory

Logics and sets

Two strategies to formally interpret natural language, inherited from Montague:

- direct: right into set theory
    - denotations (entities, functions, etc.) are elements of sets

Logics and sets

Two strategies to formally interpret natural language, inherited from Montague:

- direct: right into set theory
    - denotations (entities, functions, etc.) are elements of sets
- indirect: into a formal logic, e.g., the simply-typed λ-calculus/higher-order logic

Indirect interpretation

(1) Someone is tall.

Indirect interpretation

(1) Someone is tall.

$$[\![\textit{someone}]\!] = \lambda k.\exists x : \text{human}(x) \wedge k(x)$$

Indirect interpretation

(1) Someone is tall.

$$[\![\textit{someone}]\!] = \lambda k.\exists x : \text{human}(x) \wedge k(x)$$

$$[\![\textit{is}]\!] = \lambda x.x$$

Indirect interpretation

(1) Someone is tall.

$$
\begin{aligned}
[\![\textit{someone}]\!] &= \lambda k.\exists x : \text{human}(x) \wedge k(x) \\
[\![\textit{is}]\!] &= \lambda x.x \\
[\![\textit{tall}]\!] &= \lambda x.\text{height}(x) \geq \theta_{tall}
\end{aligned}
$$

Indirect interpretation

(1) Someone is tall.

$$
\begin{aligned}
[\![\textit{someone}]\!] &= \lambda k.\exists x : \text{human}(x) \wedge k(x) \\
[\![\textit{is}]\!] &= \lambda x.x \\
[\![\textit{tall}]\!] &= \lambda x.\text{height}(x) \geq \theta_{tall}
\end{aligned}
$$

Indirect interpretation

(1) Someone is tall.

$$\begin{aligned} [\![\textit{someone}]\!] &= \lambda k.\exists x : \text{human}(x) \wedge k(x) \\ [\![\textit{is}]\!] &= \lambda x.x \\ [\![\textit{tall}]\!] &= \lambda x.\text{height}(x) \geq \theta_{tall} \end{aligned}$$

Functional application and β-reduction:

Indirect interpretation

(1) Someone is tall.

$$\begin{aligned} [\![\textit{someone}]\!] &= \lambda k.\exists x : \text{human}(x) \wedge k(x) \\ [\![\textit{is}]\!] &= \lambda x.x \\ [\![\textit{tall}]\!] &= \lambda x.\text{height}(x) \geq \theta_{tall} \end{aligned}$$

Functional application and β-reduction:

- $[\![\textit{someone}]\!]([\![\textit{is}]\!]([\![\textit{tall}]\!])) \;\; \rightarrow_\beta \;\; \exists x : \text{human}(x) \wedge \text{height}(x) \geq \theta_{tall}$

The traditional interpretation

A λ-homomorphism

We call on a λ-homomorphism to interpret this logical language into a model…

A λ-homomorphism

We call on a λ-homomorphism to interpret this logical language into a model…

$$(\!|x|\!) = x \qquad \text{(variables)}$$

A λ-homomorphism

We call on a λ-homomorphism to interpret this logical language into a model…

$$(\!|x|\!) = x \qquad \text{(variables)}$$

$$(\!|\lambda x.M|\!) = \lambda x.(\!|M|\!) \qquad \text{(abstractions)}$$

A λ-homomorphism

We call on a λ-homomorphism to interpret this logical language into a model…

$$(\!|x|\!) = x \qquad \text{(variables)}$$

$$(\!|\lambda x.M|\!) = \lambda x.(\!|M|\!) \qquad \text{(abstractions)}$$

$$(\!|MN|\!) = (\!|M|\!)(\!|N|\!) \qquad \text{(applications)}$$

A λ-homomorphism

We call on a λ-homomorphism to interpret this logical language into a model…

$$\begin{aligned}
(\!|x|\!) &= x && \text{(variables)}\\
(\!|\lambda x.M|\!) &= \lambda x.(\!|M|\!) && \text{(abstractions)}\\
(\!|MN|\!) &= (\!|M|\!)(\!|N|\!) && \text{(applications)}\\
(\!|\langle M, N\rangle|\!) &= \langle (\!|M|\!), (\!|N|\!)\rangle && \text{(pairing)}
\end{aligned}$$

A λ-homomorphism

We call on a λ-homomorphism to interpret this logical language into a model…

$$(\!|x|\!) = x \qquad \text{(variables)}$$

$$(\!|\lambda x.M|\!) = \lambda x.(\!|M|\!) \qquad \text{(abstractions)}$$

$$(\!|MN|\!) = (\!|M|\!)(\!|N|\!) \qquad \text{(applications)}$$

$$(\!|\langle M, N\rangle|\!) = \langle (\!|M|\!), (\!|N|\!)\rangle \qquad \text{(pairing)}$$

$$(\!|M_i|\!) = (\!|M|\!)_i \qquad \text{(projection)}$$

A λ-homomorphism

We call on a λ-homomorphism to interpret this logical language into a model…

$$(\!|x|\!) = x \qquad \text{(variables)}$$

$$(\!|\lambda x.M|\!) = \lambda x.(\!|M|\!) \qquad \text{(abstractions)}$$

$$(\!|MN|\!) = (\!|M|\!)(\!|N|\!) \qquad \text{(applications)}$$

$$(\!|\langle M, N\rangle|\!) = \langle (\!|M|\!), (\!|N|\!)\rangle \qquad \text{(pairing)}$$

$$(\!|M_i|\!) = (\!|M|\!)_i \qquad \text{(projection)}$$

$$(\!|\theta_{tall}|\!) = d \qquad \text{(some real number)}$$

A λ-homomorphism

We call on a λ-homomorphism to interpret this logical language into a model…

$$(\!|x|\!) = x \qquad \text{(variables)}$$

$$(\!|\lambda x.M|\!) = \lambda x.(\!|M|\!) \qquad \text{(abstractions)}$$

$$(\!|MN|\!) = (\!|M|\!)(\!|N|\!) \qquad \text{(applications)}$$

$$(\!|\langle M, N\rangle|\!) = \langle (\!|M|\!), (\!|N|\!)\rangle \qquad \text{(pairing)}$$

$$(\!|M_i|\!) = (\!|M|\!)_i \qquad \text{(projection)}$$

$$(\!|\theta_{tall}|\!) = d \qquad \text{(some real number)}$$

$$(\!|\text{height}|\!) = \mathit{height} \qquad \text{(some function)}$$

A λ-homomorphism

We call on a λ-homomorphism to interpret this logical language into a model…

$$(\!|x|\!) = x \qquad \text{(variables)}$$

$$(\!|\lambda x.M|\!) = \lambda x.(\!|M|\!) \qquad \text{(abstractions)}$$

$$(\!|MN|\!) = (\!|M|\!)(\!|N|\!) \qquad \text{(applications)}$$

$$(\!|\langle M, N\rangle|\!) = \langle(\!|M|\!), (\!|N|\!)\rangle \qquad \text{(pairing)}$$

$$(\!|M_i|\!) = (\!|M|\!)_i \qquad \text{(projection)}$$

$$(\!|\theta_{tall}|\!) = d \qquad \text{(some real number)}$$

$$(\!|\text{height}|\!) = \mathit{height} \qquad \text{(some function)}$$

$$(\!|\text{human}|\!) = \mathit{human} \qquad \text{(some property)}$$

A λ-homomorphism

We call on a λ-homomorphism to interpret this logical language into a model...

$$\llparenthesis x \rrparenthesis = x \qquad \text{(variables)}$$

$$\llparenthesis \lambda x.M \rrparenthesis = \lambda x.\llparenthesis M \rrparenthesis \qquad \text{(abstractions)}$$

$$\llparenthesis MN \rrparenthesis = \llparenthesis M \rrparenthesis \llparenthesis N \rrparenthesis \qquad \text{(applications)}$$

$$\llparenthesis \langle M, N \rangle \rrparenthesis = \langle \llparenthesis M \rrparenthesis, \llparenthesis N \rrparenthesis \rangle \qquad \text{(pairing)}$$

$$\llparenthesis M_i \rrparenthesis = \llparenthesis M \rrparenthesis_i \qquad \text{(projection)}$$

$$\llparenthesis \theta_{tall} \rrparenthesis = d \qquad \text{(some real number)}$$

$$\llparenthesis \text{height} \rrparenthesis = \mathit{height} \qquad \text{(some function)}$$

$$\llparenthesis \text{human} \rrparenthesis = \mathit{human} \qquad \text{(some property)}$$

$$\llparenthesis (\geq) \rrparenthesis = (\geq) \qquad \text{(“less than or equal to”)}$$

A λ-homomorphism

We call on a λ-homomorphism to interpret this logical language into a model…

$$(\!|x|\!) = x \qquad \text{(variables)}$$

$$(\!|\lambda x.M|\!) = \lambda x.(\!|M|\!) \qquad \text{(abstractions)}$$

$$(\!|MN|\!) = (\!|M|\!)(\!|N|\!) \qquad \text{(applications)}$$

$$(\!|\langle M, N\rangle|\!) = \langle (\!|M|\!), (\!|N|\!)\rangle \qquad \text{(pairing)}$$

$$(\!|M_i|\!) = (\!|M|\!)_i \qquad \text{(projection)}$$

$$(\!|\theta_{tall}|\!) = d \qquad \text{(some real number)}$$

$$(\!|\text{height}|\!) = \textit{height} \qquad \text{(some function)}$$

$$(\!|\text{human}|\!) = \textit{human} \qquad \text{(some property)}$$

$$(\!|(\geq)|\!) = (\geq) \qquad \text{(“less than or equal to”)}$$

A λ-homomorphism

We call on a λ-homomorphism to interpret this logical language into a model…

$$(\!|x|\!) = x \qquad \text{(variables)}$$

$$(\!|\lambda x.M|\!) = \lambda x.(\!|M|\!) \qquad \text{(abstractions)}$$

$$(\!|MN|\!) = (\!|M|\!)(\!|N|\!) \qquad \text{(applications)}$$

$$(\!|\langle M, N\rangle|\!) = \langle (\!|M|\!), (\!|N|\!)\rangle \qquad \text{(pairing)}$$

$$(\!|M_i|\!) = (\!|M|\!)_i \qquad \text{(projection)}$$

$$(\!|\theta_{tall}|\!) = d \qquad \text{(some real number)}$$

$$(\!|\text{height}|\!) = \textit{height} \qquad \text{(some function)}$$

$$(\!|\text{human}|\!) = \textit{human} \qquad \text{(some property)}$$

$$(\!|(\geq)|\!) = (\geq) \qquad \text{(“less than or equal to”)}$$

Etc. ($\diamond$, logical constants)

Composing $(\!|\cdot|\!)$ with $[\![\cdot]\!]$

If we compose the logical interpretation with the λ-homomorphism:

Composing $(\!|\cdot|\!)$ with $[\![\cdot]\!]$

If we compose the logical interpretation with the λ-homomorphism:

- $(\!|[\![\textit{someone is tall}]\!]|\!) = \exists x : \textit{human}(x) \wedge \textit{height}(x) \geq d$

Composing $(\!|\cdot|\!)$ with $[\![\cdot]\!]$

If we compose the logical interpretation with the λ-homomorphism:

- $(\!|[\![\textit{someone is tall}]\!]|\!) = \exists x : \textit{human}(x) \wedge \textit{height}(x) \geq d$
- A truth value.

The probabilistic interpretation

Contexts

Let us assume that the non-logical constants of the logical language are finite in number and are ordered.

Contexts

Let us assume that the non-logical constants of the logical language are finite in number and are ordered.

- (1) height : $e \longrightarrow d_{tall}$ (2) human : $e \longrightarrow t$
  (3) $(\geq)$: $r \longrightarrow r \longrightarrow t$ (4) θ_{tall} : d_{tall}

Contexts

Let us assume that the non-logical constants of the logical language are finite in number and are ordered.

- (1) height : $e \longrightarrow d_{tall}$ (2) human : $e \longrightarrow t$
  (3) $(\geq)$: $r \longrightarrow r \longrightarrow t$ (4) $\theta_{tall} : d_{tall}$
- A *context* (κ) is a tuple of type $\alpha_1 \times ... \times \alpha_n$, where α_i is the type of the i^{th} constant.

Contexts

Let us assume that the non-logical constants of the logical language are finite in number and are ordered.

- (1) height : $e \longrightarrow d_{tall}$ (2) human : $e \longrightarrow t$
  (3) $(\geq) : r \longrightarrow r \longrightarrow t$ (4) $\theta_{tall} : d_{tall}$
- A *context* (κ) is a tuple of type $\alpha_1 \times ... \times \alpha_n$, where α_i is the type of the i^{th} constant.
- A context for this language would be of type
  $(e \longrightarrow d_{tall}) \times (e \longrightarrow t) \times (r \longrightarrow r \longrightarrow t) \times d_{tall}$.

A λ-homomorphism in a context

Given some context κ:

A λ-homomorphism in a context

Given some context κ:

$$(\!|x|\!)^\kappa = x \qquad \text{(variables)}$$

A λ-homomorphism in a context

Given some context κ:

$$(\!|x|\!)^\kappa = x \qquad \text{(variables)}$$

$$(\!|\lambda x.M|\!)^\kappa = \lambda x.(\!|M|\!)^\kappa \qquad \text{(abstractions)}$$

A λ-homomorphism in a context

Given some context κ:

$$\llparenthesis x \rrparenthesis^\kappa = x \qquad \text{(variables)}$$

$$\llparenthesis \lambda x.M \rrparenthesis^\kappa = \lambda x.\llparenthesis M \rrparenthesis^\kappa \qquad \text{(abstractions)}$$

$$\llparenthesis MN \rrparenthesis^\kappa = \llparenthesis M \rrparenthesis^\kappa \llparenthesis N \rrparenthesis^\kappa \qquad \text{(applications)}$$

A λ-homomorphism in a context

Given some context κ:

$$\begin{aligned}
(\!|x|\!)^\kappa &= x && \text{(variables)}\\
(\!|\lambda x.M|\!)^\kappa &= \lambda x.(\!|M|\!)^\kappa && \text{(abstractions)}\\
(\!|MN|\!)^\kappa &= (\!|M|\!)^\kappa(\!|N|\!)^\kappa && \text{(applications)}\\
(\!|\langle M, N\rangle|\!)^\kappa &= \langle (\!|M|\!)^\kappa, (\!|N|\!)^\kappa\rangle && \text{(pairing)}
\end{aligned}$$

A λ-homomorphism in a context

Given some context κ:

$$\begin{aligned}
(\!|x|\!)^\kappa &= x && \text{(variables)}\\
(\!|\lambda x.M|\!)^\kappa &= \lambda x.(\!|M|\!)^\kappa && \text{(abstractions)}\\
(\!|MN|\!)^\kappa &= (\!|M|\!)^\kappa(\!|N|\!)^\kappa && \text{(applications)}\\
(\!|\langle M, N\rangle|\!)^\kappa &= \langle(\!|M|\!)^\kappa, (\!|N|\!)^\kappa\rangle && \text{(pairing)}\\
(\!|M_i|\!)^\kappa &= (\!|M|\!)^\kappa_i && \text{(projection)}
\end{aligned}$$

A λ-homomorphism in a context

Given some context κ:

$$
\begin{aligned}
(\!|x|\!)^\kappa &= x && \text{(variables)}\\
(\!|\lambda x.M|\!)^\kappa &= \lambda x.(\!|M|\!)^\kappa && \text{(abstractions)}\\
(\!|MN|\!)^\kappa &= (\!|M|\!)^\kappa(\!|N|\!)^\kappa && \text{(applications)}\\
(\!|\langle M, N\rangle|\!)^\kappa &= \langle(\!|M|\!)^\kappa, (\!|N|\!)^\kappa\rangle && \text{(pairing)}\\
(\!|M_i|\!)^\kappa &= (\!|M|\!)_i^\kappa && \text{(projection)}\\
(\!|c_i|\!)^\kappa &= \kappa_i && (c_i \text{ is the } i^{th} \text{ constant})
\end{aligned}
$$

A λ-homomorphism in a context

Given some context κ:

$$
\begin{aligned}
(\!|x|\!)^\kappa &= x && \text{(variables)}\\
(\!|\lambda x.M|\!)^\kappa &= \lambda x.(\!|M|\!)^\kappa && \text{(abstractions)}\\
(\!|MN|\!)^\kappa &= (\!|M|\!)^\kappa (\!|N|\!)^\kappa && \text{(applications)}\\
(\!|\langle M, N\rangle|\!)^\kappa &= \langle (\!|M|\!)^\kappa, (\!|N|\!)^\kappa \rangle && \text{(pairing)}\\
(\!|M_i|\!)^\kappa &= (\!|M|\!)^\kappa_i && \text{(projection)}\\
(\!|c_i|\!)^\kappa &= \kappa_i && \text{(c_i is the i^{th} constant)}
\end{aligned}
$$

A λ-homomorphism in a context

Given some context κ:

$$\begin{aligned}
(\!|x|\!)^\kappa &= x && \text{(variables)}\\
(\!|\lambda x.M|\!)^\kappa &= \lambda x.(\!|M|\!)^\kappa && \text{(abstractions)}\\
(\!|MN|\!)^\kappa &= (\!|M|\!)^\kappa (\!|N|\!)^\kappa && \text{(applications)}\\
(\!|\langle M, N\rangle|\!)^\kappa &= \langle (\!|M|\!)^\kappa, (\!|N|\!)^\kappa \rangle && \text{(pairing)}\\
(\!|M_i|\!)^\kappa &= (\!|M|\!)^\kappa_i && \text{(projection)}\\
(\!|c_i|\!)^\kappa &= \kappa_i && (c_i \text{ is the } i^{th} \text{ constant})
\end{aligned}$$

Etc. ($\diamond$, logical constants)

Composing $(\!|\cdot|\!)^\kappa$ with $[\![\cdot]\!]$

If we compose the logical interpretation with the λ-homomorphism in some context κ:

Composing $(\!|\cdot|\!)^\kappa$ with $[\![\cdot]\!]$

If we compose the logical interpretation with the λ-homomorphism in some context κ:

- $(\!|[\![\textit{someone is tall}]\!]|\!)^\kappa = \exists x : \kappa_2(x) \wedge \kappa_3(\kappa_1(x))(\kappa_4)$

Composing $(\!|\cdot|\!)^\kappa$ with $[\![\cdot]\!]$

If we compose the logical interpretation with the λ-homomorphism in some context κ:

- $(\!|[\![\textit{someone is tall}]\!]|\!)^\kappa = \exists x : \kappa_2(x) \land \kappa_3(\kappa_1(x))(\kappa_4)$
- A truth value.

Composing $(\!|\cdot|\!)^\kappa$ with $[\![\cdot]\!]$

If we compose the logical interpretation with the λ-homomorphism in some context κ:

- $(\!|[\![\textit{someone is tall}]\!]|\!)^\kappa = \exists x : \kappa_2(x) \land \kappa_3(\kappa_1(x))(\kappa_4)$
- A truth value.

Composing $(\!|\cdot|\!)^\kappa$ with $[\![\cdot]\!]$

If we compose the logical interpretation with the λ-homomorphism in some context κ:

- $(\!|[\![\textit{someone is tall}]\!]|\!)^\kappa = \exists x : \kappa_2(x) \wedge \kappa_3(\kappa_1(x))(\kappa_4)$
- A truth value.

For example, say $\kappa = \langle \textit{height}, \textit{human}, (\geq), d \rangle$:

Composing $(\!|\cdot|\!)^\kappa$ with $[\![\cdot]\!]$

If we compose the logical interpretation with the λ-homomorphism in some context κ:

- $(\!|[\![\textit{someone is tall}]\!]|\!)^\kappa = \exists x : \kappa_2(x) \wedge \kappa_3(\kappa_1(x))(\kappa_4)$
- A truth value.

For example, say $\kappa = \langle \textit{height}, \textit{human}, (\geq), d \rangle$:

- $(\!|[\![\textit{someone is tall}]\!]|\!)^\kappa = \exists x : \textit{human}(x) \wedge \textit{height}(x) \geq d$

Composing $(\!|\cdot|\!)^\kappa$ with $[\![\cdot]\!]$

If we compose the logical interpretation with the λ-homomorphism in some context κ:

- $(\!|[\![\textit{someone is tall}]\!]|\!)^\kappa = \exists x : \kappa_2(x) \land \kappa_3(\kappa_1(x))(\kappa_4)$
- A truth value.

For example, say $\kappa = \langle \textit{height}, \textit{human}, (\geq), d \rangle$:

- $(\!|[\![\textit{someone is tall}]\!]|\!)^\kappa = \exists x : \textit{human}(x) \land \textit{height}(x) \geq d$

Composing $(\!|\cdot|\!)^\kappa$ with $[\![\cdot]\!]$

If we compose the logical interpretation with the λ-homomorphism in some context κ:

- $(\!|[\![\textit{someone is tall}]\!]|\!)^\kappa = \exists x : \kappa_2(x) \wedge \kappa_3(\kappa_1(x))(\kappa_4)$
- A truth value.

For example, say $\kappa = \langle \textit{height}, \textit{human}, (\geq), d \rangle$:

- $(\!|[\![\textit{someone is tall}]\!]|\!)^\kappa = \exists x : \textit{human}(x) \wedge \textit{height}(x) \geq d$

Goal: allow the context to be a random variable.

Probabilistic programs

Definition

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

Definition

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

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

Definition

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/integrating f over the possible values x of type α, weighting each $f(x)$ by the probability of x.

Definition

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/integrating f over the possible values x of type α, weighting each $f(x)$ by the probability of x.
- E.g., $\mathcal{N}(\mu, \sigma) : (d_{tall} \to r) \to r$

Definition

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/integrating f over the possible values x of type α, weighting each $f(x)$ by the probability of x.
- E.g., $\mathcal{N}(\mu, \sigma) : (d_{tall} \to r) \to r$
    - Represents a normal distribution with mean μ and standard deviation σ.

Definition

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/integrating f over the possible values x of type α, weighting each $f(x)$ by the probability of x.
- E.g., $\mathcal{N}(\mu, \sigma) : (d_{tall} \to r) \to r$
    - Represents a normal distribution with mean μ and standard deviation σ.
    - $\mathcal{N}(\mu, \sigma)(f) = \int_{-\infty}^{\infty} \text{PDF}_{\mathcal{N}(\mu,\sigma)}(x) * f(x) dx$

Some nice things about probabilistic programs (pt. 1)

You can turn any value (of any type α) into a trivial probabilistic program that returns just that value:

Some nice things about probabilistic programs (pt. 1)

You can turn any value (of any type α) into a trivial probabilistic program that returns just that value:

$$\eta : \alpha \to (\alpha \to r) \to r \qquad \text{('return')}$$

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

Some nice things about probabilistic programs (pt. 1)

You can turn any value (of any type α) into a trivial probabilistic program that returns just that value:

$$\eta : \alpha \to (\alpha \to r) \to r \qquad \text{(`return')}$$
$$\eta(a) = \lambda f.f(a)$$

E.g., $\eta(jp) = \lambda f.f(jp)$:

Some nice things about probabilistic programs (pt. 1)

You can turn any value (of any type α) into a trivial probabilistic program that returns just that value:

$$\eta : \alpha \to (\alpha \to r) \to r \qquad \text{('return')}$$
$$\eta(a) = \lambda f.f(a)$$

E.g., $\eta(jp) = \lambda f.f(jp)$:

- The probabilistic program that returns Jean-Philippe with a probability of 1.

Some nice things about probabilistic programs (pt. 2)

You can pass the value returned by a probabilistic program m to a function k from values to probabilistic programs, in order to make a bigger, sequenced probabilistic program.

Some nice things about probabilistic programs (pt. 2)

You can pass the value returned by a probabilistic program *m* to a function *k* from values to probabilistic programs, in order to make a bigger, sequenced probabilistic program.

$$
\begin{aligned}
(\star) : ((\alpha \to r) \to r) & & \text{('bind')} \\
\to (\alpha \to (\beta \to r) \to r) & & \\
\to (\beta \to r) \to r & & \\
m \star k = \lambda f.m(\lambda x.k(x)(f)) & &
\end{aligned}
$$

Some nice things about probabilistic programs (pt. 2)

You can pass the value returned by a probabilistic program m to a function k from values to probabilistic programs, in order to make a bigger, sequenced probabilistic program.

$$\begin{aligned}(\star) : ((\alpha \to r) \to r) \qquad\qquad & \text{(`bind')} \\ \to (\alpha \to (\beta \to r) \to r) & \\ \to (\beta \to r) \to r & \\ m \star k = \lambda f.m(\lambda x.k(x)(f)) &\end{aligned}$$

"Run m, computing x. Then feed x to k."

What this all means

Probabilistic programs form a monad.

Monads

Operators

$$\eta : \alpha \to M\alpha$$
$$(\star) : M\alpha \to (\alpha \to M\beta) \to M\beta$$

Monads

Operators

$$\eta : \alpha \to M\alpha$$
$$(\star) : M\alpha \to (\alpha \to M\beta) \to M\beta$$

Laws on terms

$$\eta(v) \star k = k(v) \qquad \text{(Left Identity)}$$
$$m \star \eta = m \qquad \text{(Right Identity)}$$
$$(m \star n) \star o = m \star (\lambda x.n(x) \star o) \qquad \text{(Associativity)}$$

The continuation monad

The monad formed by probabilistic programs is a version of the *continuation monad*, i.e., where the result of the continuation is an r:

$$\eta : \alpha \to (\alpha \to r) \to r$$

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

$$\begin{aligned}(\star) : &((\alpha \to r) \to r) \\ &\to (\alpha \to (\beta \to r) \to r) \\ &\to (\beta \to r) \to r\end{aligned}$$

$$m \star k = \lambda c.m(\lambda x.k(x)(c))$$

The continuation monad

The monad formed by probabilistic programs is a version of the *continuation monad*, i.e., where the result of the continuation is an r:

$$\eta : \alpha \to (\alpha \to r) \to r$$

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

$$\begin{aligned}(\star) : ((\alpha \to r) \to r) \\ \to (\alpha \to (\beta \to r) \to r) \\ \to (\beta \to r) \to r\end{aligned}$$

$$m \star k = \lambda c.m(\lambda x.k(x)(c))$$

The continuations are just projection functions.

Building probabilistic programs

We may now build probabilistic programs that return *contexts*.

Building probabilistic programs

We may now build probabilistic programs that return *contexts*.

- If a context is of type $\alpha_1 \times ... \times \alpha_n$, then we seek a probabilistic program K of type $(\alpha_1 \times ... \times \alpha_n \to r) \to r$.

Building probabilistic programs

We may now build probabilistic programs that return *contexts*.

- If a context is of type $\alpha_1 \times ... \times \alpha_n$, then we seek a probabilistic program K of type $(\alpha_1 \times ... \times \alpha_n \to r) \to r$.
- Then, for a sentence ϕ in the logical language, we may do:

$$K \star \lambda\kappa.\eta((\!|\phi|\!)^\kappa) : (t \to r) \to r$$

Computing probabilities

Once we have a probabilistic program of type $(t \to r) \to r$, we may compute a probability from it:

Computing probabilities

Once we have a probabilistic program of type $(t \to r) \to r$, we may compute a probability from it:

$$P : ((t \to r) \to r) \to r$$

Computing probabilities

Once we have a probabilistic program of type $(t \to r) \to r$, we may compute a probability from it:

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

Computing probabilities

Once we have a probabilistic program of type $(t \to r) \to r$, we may compute a probability from it:

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

Computing probabilities

Once we have a probabilistic program of type $(t \rightarrow r) \rightarrow r$, we may compute a probability from it:

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

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

Computing probabilities

Once we have a probabilistic program of type $(t \to r) \to r$, we may compute a probability from it:

$$P : ((t \to r) \to r) \to r$$

$$P(p) = \frac{p(\mathbb{1})}{p(\lambda b.1)}$$

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

Computing probabilities

Once we have a probabilistic program of type $(t \to r) \to r$, we may compute a probability from it:

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

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

Computing probabilities

Once we have a probabilistic program of type $(t \to r) \to r$, we may compute a probability from it:

$$P : ((t \to r) \to r) \to r$$

$$P(p) = \frac{p(\mathbb{1})}{p(\lambda b.1)}$$

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

Computing probabilities

Once we have a probabilistic program of type $(t \to r) \to r$, we may compute a probability from it:

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

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

Computing probabilities

Once we have a probabilistic program of type $(t \to r) \to r$, we may compute a probability from it:

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

- $\mathbb{1} : t \to r$ is an indicator function:
  - $\mathbb{1}(\top) = 1$
  - $\mathbb{1}(\bot) = 0$
- In the above, it picks out the mass assigned to $\top$.
- $\lambda b.1$ picks out the total mass (assigned to $\top$ and $\bot$).
- So, $P(p)$ is the probability that p returns $\top$.

An example

(1) Someone is tall.

An example

(1) Someone is tall.

Say our constants are:

1. height : $e \rightarrow d_{tall}$
2. human : $e \rightarrow t$
3. $(\geq) : r \rightarrow r \rightarrow t$
4. $\theta_{tall} : d_{tall}$

An example

(1) Someone is tall.

Say our constants are:

1. height : $e \to d_{tall}$
2. human : $e \to t$
3. $(\geq) : r \to r \to t$
4. $\theta_{tall} : d_{tall}$

Define K as:

$$K = \mathcal{N}(72, 3) \star \lambda d.\eta(height, human, (\geq), d)$$

An example (cont'd)

$$K \star \lambda\kappa.\eta(\llparenthesis \exists x : \text{human}(x) \wedge \text{height}(x) \geq \theta_{tall} \rrparenthesis^{\kappa})$$

An example (cont'd)

$$
\begin{aligned}
& K \star \lambda\kappa.\eta(\llparenthesis \exists x : \text{human}(x) \wedge \text{height}(x) \geq \theta_{tall} \rrparenthesis^{\kappa}) \\
= \; & K \star \lambda\kappa.\eta(\exists x : \llparenthesis \text{human} \rrparenthesis^{\kappa}(x) \wedge \llparenthesis (\geq) \rrparenthesis^{\kappa}(\llparenthesis \text{height} \rrparenthesis^{\kappa}(x))(\llparenthesis \theta_{tall} \rrparenthesis^{\kappa}))
\end{aligned}
$$

An example (cont'd)

$$
\begin{aligned}
& K \star \lambda\kappa.\eta(\llparenthesis \exists x : \text{human}(x) \wedge \text{height}(x) \geq \theta_{tall} \rrparenthesis^{\kappa}) \\
= \ & K \star \lambda\kappa.\eta(\exists x : \llparenthesis \text{human} \rrparenthesis^{\kappa}(x) \wedge \llparenthesis (\geq) \rrparenthesis^{\kappa}(\llparenthesis \text{height} \rrparenthesis^{\kappa}(x))(\llparenthesis \theta_{tall} \rrparenthesis^{\kappa})) \\
= \ & \mathcal{N}(72, 3) \star \lambda d.\eta(\exists x : \mathit{human}(x) \wedge \mathit{height}(x) \geq d)
\end{aligned}
$$

An example (cont'd)

$$
\begin{aligned}
& K \star \lambda\kappa.\eta(⦇\exists x : \text{human}(x) \wedge \text{height}(x) \geq \theta_{tall}⦈^{\kappa}) \\
= \; & K \star \lambda\kappa.\eta(\exists x : ⦇\text{human}⦈^{\kappa}(x) \wedge ⦇(\geq)⦈^{\kappa}(⦇\text{height}⦈^{\kappa}(x))(⦇\theta_{tall}⦈^{\kappa})) \\
= \; & \mathcal{N}(72, 3) \star \lambda d.\eta(\exists x : \mathit{human}(x) \wedge \mathit{height}(x) \geq d) \\
= \; & \lambda f.\, \mathcal{N}(72, 3)(\lambda d.f(\exists x : \mathit{human}(x) \wedge \mathit{height}(x) \geq d))
\end{aligned}
$$

An example (cont'd)

$$\begin{aligned}
& K \star \lambda\kappa.\eta(\llparenthesis \exists x : \text{human}(x) \wedge \text{height}(x) \geq \theta_{tall} \rrparenthesis^{\kappa}) \\
= \; & K \star \lambda\kappa.\eta(\exists x : \llparenthesis \text{human} \rrparenthesis^{\kappa}(x) \wedge \llparenthesis (\geq) \rrparenthesis^{\kappa}(\llparenthesis \text{height} \rrparenthesis^{\kappa}(x))(\llparenthesis \theta_{tall} \rrparenthesis^{\kappa})) \\
= \; & \mathcal{N}(72, 3) \star \lambda d.\eta(\exists x : \mathit{human}(x) \wedge \mathit{height}(x) \geq d) \\
= \; & \lambda f.\, \mathcal{N}(72, 3)(\lambda d.f(\exists x : \mathit{human}(x) \wedge \mathit{height}(x) \geq d))
\end{aligned}$$

An example (cont'd)

$$
\begin{aligned}
& K \star \lambda\kappa.\eta(\llparenthesis \exists x : \text{human}(x) \wedge \text{height}(x) \geq \theta_{tall} \rrparenthesis^{\kappa}) \\
= \; & K \star \lambda\kappa.\eta(\exists x : \llparenthesis \text{human} \rrparenthesis^{\kappa}(x) \wedge \llparenthesis (\geq) \rrparenthesis^{\kappa}(\llparenthesis \text{height} \rrparenthesis^{\kappa}(x))(\llparenthesis \theta_{tall} \rrparenthesis^{\kappa})) \\
= \; & \mathcal{N}(72, 3) \star \lambda d.\eta(\exists x : \mathit{human}(x) \wedge \mathit{height}(x) \geq d) \\
= \; & \lambda f.\mathcal{N}(72, 3)(\lambda d.f(\exists x : \mathit{human}(x) \wedge \mathit{height}(x) \geq d))
\end{aligned}
$$

Computing a probability:

An example (cont'd)

$$
\begin{aligned}
& K \star \lambda\kappa.\eta(\llparenthesis \exists x : \text{human}(x) \wedge \text{height}(x) \geq \theta_{tall} \rrparenthesis^{\kappa}) \\
= \; & K \star \lambda\kappa.\eta(\exists x : \llparenthesis \text{human} \rrparenthesis^{\kappa}(x) \wedge \llparenthesis (\geq) \rrparenthesis^{\kappa}(\llparenthesis \text{height} \rrparenthesis^{\kappa}(x))(\llparenthesis \theta_{tall} \rrparenthesis^{\kappa})) \\
= \; & \mathcal{N}(72, 3) \star \lambda d.\eta(\exists x : \mathit{human}(x) \wedge \mathit{height}(x) \geq d) \\
= \; & \lambda f.\mathcal{N}(72, 3)(\lambda d.f(\exists x : \mathit{human}(x) \wedge \mathit{height}(x) \geq d))
\end{aligned}
$$

Computing a probability:

$$
\frac{\mathcal{N}(72, 3)(\lambda d.\mathbb{1}(\exists x : \mathit{human}(x) \wedge \mathit{height}(x) \geq d))}{\mathcal{N}(72, 3)(\lambda d.1)}
$$

An example (cont'd)

$$\begin{aligned}
& K \star \lambda\kappa.\eta(\llparenthesis \exists x : \text{human}(x) \wedge \text{height}(x) \geq \theta_{tall} \rrparenthesis^{\kappa}) \\
=\ & K \star \lambda\kappa.\eta(\exists x : \llparenthesis \text{human} \rrparenthesis^{\kappa}(x) \wedge \llparenthesis (\geq) \rrparenthesis^{\kappa}(\llparenthesis \text{height} \rrparenthesis^{\kappa}(x))(\llparenthesis \theta_{tall} \rrparenthesis^{\kappa})) \\
=\ & \mathcal{N}(72, 3) \star \lambda d.\eta(\exists x : \mathit{human}(x) \wedge \mathit{height}(x) \geq d) \\
=\ & \lambda f.\mathcal{N}(72, 3)(\lambda d.f(\exists x : \mathit{human}(x) \wedge \mathit{height}(x) \geq d))
\end{aligned}$$

Computing a probability:

$$\begin{aligned}
& \frac{\mathcal{N}(72, 3)(\lambda d.\mathbb{1}(\exists x : \mathit{human}(x) \wedge \mathit{height}(x) \geq d))}{\mathcal{N}(72, 3)(\lambda d.1)} \\
=\ & \mathcal{N}(72, 3)(\lambda d.\mathbb{1}(\exists x : \mathit{human}(x) \wedge \mathit{height}(x) \geq d))
\end{aligned}$$

An example (cont'd)

$$\begin{aligned} & K \star \lambda\kappa.\eta(\llparenthesis \exists x : \text{human}(x) \wedge \text{height}(x) \geq \theta_{tall} \rrparenthesis^{\kappa}) \\ = \ & K \star \lambda\kappa.\eta(\exists x : \llparenthesis\text{human}\rrparenthesis^{\kappa}(x) \wedge \llparenthesis(\geq)\rrparenthesis^{\kappa}(\llparenthesis\text{height}\rrparenthesis^{\kappa}(x))(\llparenthesis\theta_{tall}\rrparenthesis^{\kappa})) \\ = \ & \mathcal{N}(72, 3) \star \lambda d.\eta(\exists x : \mathit{human}(x) \wedge \mathit{height}(x) \geq d) \\ = \ & \lambda f.\mathcal{N}(72, 3)(\lambda d.f(\exists x : \mathit{human}(x) \wedge \mathit{height}(x) \geq d)) \end{aligned}$$

Computing a probability:

$$\begin{aligned} & \frac{\mathcal{N}(72, 3)(\lambda d.\mathbb{1}(\exists x : \mathit{human}(x) \wedge \mathit{height}(x) \geq d))}{\mathcal{N}(72, 3)(\lambda d.1)} \\ = \ & \mathcal{N}(72, 3)(\lambda d.\mathbb{1}(\exists x : \mathit{human}(x) \wedge \mathit{height}(x) \geq d)) \end{aligned}$$

An example (cont'd)

$$
\begin{aligned}
& K \star \lambda\kappa.\eta(\llparenthesis \exists x : \text{human}(x) \wedge \text{height}(x) \geq \theta_{tall} \rrparenthesis^{\kappa}) \\
= \; & K \star \lambda\kappa.\eta(\exists x : \llparenthesis \text{human} \rrparenthesis^{\kappa}(x) \wedge \llparenthesis (\geq) \rrparenthesis^{\kappa}(\llparenthesis \text{height} \rrparenthesis^{\kappa}(x))(\llparenthesis \theta_{tall} \rrparenthesis^{\kappa})) \\
= \; & \mathcal{N}(72, 3) \star \lambda d.\eta(\exists x : \mathit{human}(x) \wedge \mathit{height}(x) \geq d) \\
= \; & \lambda f.\, \mathcal{N}(72, 3)(\lambda d.f(\exists x : \mathit{human}(x) \wedge \mathit{height}(x) \geq d))
\end{aligned}
$$

Computing a probability:

$$
\begin{aligned}
& \frac{\mathcal{N}(72, 3)(\lambda d.\mathbb{1}(\exists x : \mathit{human}(x) \wedge \mathit{height}(x) \geq d))}{\mathcal{N}(72, 3)(\lambda d.1)} \\
= \; & \mathcal{N}(72, 3)(\lambda d.\mathbb{1}(\exists x : \mathit{human}(x) \wedge \mathit{height}(x) \geq d))
\end{aligned}
$$

Probability is the mass of $\mathcal{N}(72, 3)$ less than or equal to the height of the tallest human.

Bayesian inference

Observing a premise

$$observe : t \rightarrow (\diamond \rightarrow r) \rightarrow r$$

$$observe(\phi)(f) = \mathbb{1}(\phi) * f(\diamond)$$

Semantic learning

Semantic learning is a matter of updating distributions over contexts (i.e., probabilistic programs returning contexts).

Semantic learning

Semantic learning is a matter of updating distributions over contexts (i.e., probabilistic programs returning contexts).

- Given some initial distribution K_0:

Semantic learning

Semantic learning is a matter of updating distributions over contexts (i.e., probabilistic programs returning contexts).

- Given some initial distribution K_0:

Semantic learning

Semantic learning is a matter of updating distributions over contexts (i.e., probabilistic programs returning contexts).

- Given some initial distribution K_0:

$$K_1 = K_0 \star \lambda\kappa.observe(\phi_1) \star \lambda\diamond. \ldots observe(\phi_n) \star \lambda\diamond.\eta(\kappa)$$

Semantic learning: an example

Say we have the constants:

- c, m, a, v : e
- height : $e \rightarrow d_{tall}$
- $(\geq) : r \rightarrow r \rightarrow t$
- θ_{tall}

Semantic learning: an example

Say we have the constants:

- c, m, a, v : e
- height : $e \rightarrow d_{tall}$
- $(\geq) : r \rightarrow r \rightarrow t$
- θ_{tall}

Say we start out with the initial context:

$$K_0 = \mathcal{N}(68, 3) \star \lambda d.\eta(c, m, a, v, height, (\geq), d)$$

Semantic learning: an example

Say we have the constants:

- c, m, a, v : e
- height : $e \rightarrow d_{tall}$
- $(\geq) : r \rightarrow r \rightarrow t$
- θ_{tall}

Say we start out with the initial context:

$$K_0 = \mathcal{N}(68, 3) \star \lambda d.\eta(c, m, a, v, height, (\geq), d)$$

Say we know:

$$height(c) = 65 \qquad height(m) = 67 \qquad height(a) = 72$$

Semantic learning: an example

Say we have the constants:

- c, m, a, v : e
- height : $e \rightarrow d_{tall}$
- $(\geq) : r \rightarrow r \rightarrow t$
- θ_{tall}

Say we start out with the initial context:

$$K_0 = \mathcal{N}(68, 3) \star \lambda d.\eta(c, m, a, v, height, (\geq), d)$$

Say we know:

$$height(c) = 65 \qquad height(m) = 67 \qquad height(a) = 72$$

Someone utters, "Camilla isn't tall. Matt isn't tall. Anna is tall."

Semantic learning

$$
\begin{aligned}
K_1 = K_0 & \\
& \star\ \lambda\kappa.observe(\llparenthesis\neg \text{height}(\text{c}) \geq \theta_{tall}\rrparenthesis^{\kappa}) \\
& \star\ \lambda\diamond.observe(\llparenthesis\neg \text{height}(\text{m}) \geq \theta_{tall}\rrparenthesis^{\kappa}) \\
& \star\ \lambda\diamond.observe(\llparenthesis \text{height}(\text{a}) \geq \theta_{tall}\rrparenthesis^{\kappa}) \\
& \star\ \lambda\diamond.\eta(\kappa)
\end{aligned}
$$

$$
\begin{aligned}
K_1 = K_0 & \\
& \star\ \lambda\kappa.\mathit{observe}(⦇\neg \mathrm{height}(\mathrm{c}) \geq \theta_{tall}⦈^{\kappa}) \\
& \star\ \lambda\diamond.\mathit{observe}(⦇\neg \mathrm{height}(\mathrm{m}) \geq \theta_{tall}⦈^{\kappa}) \\
& \star\ \lambda\diamond.\mathit{observe}(⦇\mathrm{height}(\mathrm{a}) \geq \theta_{tall}⦈^{\kappa}) \\
& \star\ \lambda\diamond.\eta(\kappa) \\
= \mathcal{N}(68, 3) & \qquad\qquad \text{(by Associativity and Left Identity)} \\
& \star\ \lambda d.\mathit{observe}(\neg 65 \geq d) \\
& \star\ \lambda\diamond.\mathit{observe}(\neg 67 \geq d) \\
& \star\ \lambda\diamond.\mathit{observe}(72 \geq d) \\
& \star\ \lambda\diamond.\eta(c, m, a, v, \mathit{height}, (\geq), d)
\end{aligned}
$$

Semantic learning

$$
\begin{aligned}
K_1 = K_0 \\
&\star \lambda\kappa.observe(\llparenthesis\neg\text{height}(\text{c}) \geq \theta_{tall}\rrparenthesis^{\kappa}) \\
&\star \lambda\diamond.observe(\llparenthesis\neg\text{height}(\text{m}) \geq \theta_{tall}\rrparenthesis^{\kappa}) \\
&\star \lambda\diamond.observe(\llparenthesis\text{height}(\text{a}) \geq \theta_{tall}\rrparenthesis^{\kappa}) \\
&\star \lambda\diamond.\eta(\kappa) \\
= \mathcal{N}(68, 3) & \qquad\qquad \text{(by Associativity and Left Identity)} \\
&\star \lambda d.observe(\neg 65 \geq d) \\
&\star \lambda\diamond.observe(\neg 67 \geq d) \\
&\star \lambda\diamond.observe(72 \geq d) \\
&\star \lambda\diamond.\eta(c, m, a, v, height, (\geq), d)
\end{aligned}
$$

Semantic learning

$$
\begin{aligned}
K_1 = K_0 \\
&\star \lambda\kappa.\mathit{observe}(\llparenthesis\neg\text{height}(\text{c}) \geq \theta_{tall}\rrparenthesis^{\kappa}) \\
&\star \lambda\diamond.\mathit{observe}(\llparenthesis\neg\text{height}(\text{m}) \geq \theta_{tall}\rrparenthesis^{\kappa}) \\
&\star \lambda\diamond.\mathit{observe}(\llparenthesis\text{height}(\text{a}) \geq \theta_{tall}\rrparenthesis^{\kappa}) \\
&\star \lambda\diamond.\eta(\kappa) \\
= \mathcal{N}(68, 3) & \qquad\qquad \text{(by Associativity and Left Identity)} \\
&\star \lambda d.\mathit{observe}(\neg 65 \geq d) \\
&\star \lambda\diamond.\mathit{observe}(\neg 67 \geq d) \\
&\star \lambda\diamond.\mathit{observe}(72 \geq d) \\
&\star \lambda\diamond.\eta(c, m, a, v, \mathit{height}, (\geq), d)
\end{aligned}
$$

We've pared down the distribution to the interval $(67, 72]$.

Comparing K_0 and K_1

(2) Vlad is tall.

Comparing K_0 and K_1

(2) Vlad is tall.

Say $height(v) = 68$.

Comparing K_0 and K_1

(2) Vlad is tall.

Say $height(v) = 68$.

Then:

$$\frac{K_0(\lambda\kappa.\mathbb{1}((\!|\text{height}(\text{v}) \geq \theta_{tall}|\!)^{\kappa}))}{K_0(\lambda\kappa.1)} = 0.5$$

Comparing K_0 and K_1

(2) Vlad is tall.

Say $height(v) = 68$.

Then:

$$\frac{K_0(\lambda\kappa.\mathbb{1}(\llparenthesis \text{height(v)} \geq \theta_{tall} \rrparenthesis^{\kappa}))}{K_0(\lambda\kappa.1)} = 0.5$$

$$\frac{K_1(\lambda\kappa.\mathbb{1}(\llparenthesis \text{height(v)} \geq \theta_{tall} \rrparenthesis^{\kappa}))}{K_1(\lambda\kappa.1)} \approx 0.24$$

RSA models: a popular application of probabilistic semantics.

RSA models: a popular application of probabilistic semantics. The basic idea:

RSA models: a popular application of probabilistic semantics. The basic idea:

- The RSA framework models a pragmatic listener, L_1…

RSA

RSA models: a popular application of probabilistic semantics. The basic idea:

- The RSA framework models a pragmatic listener, L_1…
- …who infers a distribution over meanings m from an utterance u, based on the probability that a pragmatic speaker, S_1, would make the utterance u to convey m.

RSA models: a popular application of probabilistic semantics. The basic idea:

- The RSA framework models a pragmatic listener, L_1…
- …who infers a distribution over meanings *m* from an utterance *u*, based on the probability that a pragmatic speaker, S_1, would make the utterance *u* to convey *m*.
- Given a meaning *m*, the probability that S_1 would make the utterance *u* to convey *m* is related to the probability that a literal listener, L_0, would infer *m*, given a literal interpretation of *u*.

RSA: Lassiter and Goodman (2013)

$$P_{L_1}(h, d_{tall} \mid \text{'}Vlad\ is\ tall\text{'}) \propto P_{S_1}(\text{'}Vlad\ is\ tall\text{'} \mid h, d_{tall}) * P_{L_1}(h) \quad (L_1)$$

$$(S_1)$$

$$(L_0)$$

RSA: Lassiter and Goodman (2013)

$$P_{L_1}(h, d_{tall} \mid \text{'Vlad is tall'}) \propto P_{S_1}(\text{'Vlad is tall'} \mid h, d_{tall}) * P_{L_1}(h) \quad (L_1)$$

$$P_{S_1}(u \mid h, d_{tall}) \propto (P_{L_0}(h \mid u, d_{tall}) * e^{-C(u)})^{\alpha} \quad (S_1)$$

$$(L_0)$$

RSA: Lassiter and Goodman (2013)

$$P_{L_1}(h, d_{tall} \mid \text{‘Vlad is tall’}) \propto P_{S_1}(\text{‘Vlad is tall’} \mid h, d_{tall}) * P_{L_1}(h) \quad (L_1)$$

$$P_{S_1}(u \mid h, d_{tall}) \propto (P_{L_0}(h \mid u, d_{tall}) * e^{-C(u)})^{\alpha} \quad (S_1)$$

$$P_{L_0}(h \mid u, d_{tall}) = P_{L_0}(h \mid [\![u]\!]^{d_{tall}} = \top) \quad (L_0)$$

RSA: more generally

$$P_{L_1}(w, \theta \mid u_0) = \frac{P_{S_1}(u_0 \mid w, \theta) * P_{L_1}(w, \theta)}{\int_{w' \in W} \int_{\theta' \in \Theta} P_{S_1}(u_0 \mid w', \theta') * P_{L_1}(w', \theta') d\theta' dw'} \quad (L_1)$$

$$(S_1)$$

$$(L_0)$$

RSA: more generally

$$P_{L_1}(w, \theta \mid u_0) = \frac{P_{S_1}(u_0 \mid w, \theta) * P_{L_1}(w, \theta)}{\int_{w' \in W} \int_{\theta' \in \Theta} P_{S_1}(u_0 \mid w', \theta') * P_{L_1}(w', \theta') d\theta' dw'} \quad (L_1)$$

$$P_{S_1}(u \mid w, \theta) = \frac{(P_{L_0}(w \mid u, \theta) * e^{-C(u)})^{\alpha}}{\Sigma_{u' \in U}(P_{L_0}(w \mid u', \theta) * e^{-C(u')})^{\alpha}} \quad (S_1)$$

$$(L_0)$$

RSA: more generally

$$P_{L_1}(w, \theta \mid u_0) = \frac{P_{S_1}(u_0 \mid w, \theta) * P_{L_1}(w, \theta)}{\int_{w' \in W} \int_{\theta' \in \Theta} P_{S_1}(u_0 \mid w', \theta') * P_{L_1}(w', \theta') d\theta' dw'} \quad (L_1)$$

$$P_{S_1}(u \mid w, \theta) = \frac{(P_{L_0}(w \mid u, \theta) * e^{-C(u)})^{\alpha}}{\Sigma_{u' \in U}(P_{L_0}(w \mid u', \theta) * e^{-C(u')})^{\alpha}} \quad (S_1)$$

$$P_{L_0}(w \mid u, \theta) = P_{L_0}(w \mid [\![u]\!]^{\theta} = \top) \quad (L_0)$$

Factoring by a weight

Recall *observe*:

Factoring by a weight

Recall *observe*:

$$\begin{aligned} \mathit{observe} &: t \to (\diamond \to r) \to r \\ \mathit{observe}(\phi)(f) &= \mathbb{1}(\phi) * f(\diamond) \end{aligned}$$

Factoring by a weight

Recall *observe*:

$$observe : t \to (\diamond \to r) \to r$$
$$observe(\phi)(f) = \mathbb{1}(\phi) * f(\diamond)$$

A useful generalization:

Factoring by a weight

Recall *observe*:

$$observe : t \to (\diamond \to r) \to r$$
$$observe(\phi)(f) = \mathbb{1}(\phi) * f(\diamond)$$

A useful generalization:

$$factor : r \to (\diamond \to r) \to r$$
$$factor(x)(f) = x * f(\diamond)$$

Another preliminary

We would also like to be able to obtain a probability density function from a distribution.

Another preliminary

We would also like to be able to obtain a probability density function from a distribution.

$$\mathsf{PDF}_{(\cdot)} : ((\alpha \to r) \to r) \to \alpha \to r$$

Another preliminary

We would also like to be able to obtain a probability density function from a distribution.

$$\mathrm{PDF}_{(\cdot)} : ((\alpha \to r) \to r) \to \alpha \to r$$

The discrete case:

$$\mathrm{PDF}_p = \lambda x.P(p \star \lambda y.\eta(y = x))$$

Another preliminary

We would also like to be able to obtain a probability density function from a distribution.

$$\mathsf{PDF}_{(\cdot)} : ((\alpha \to r) \to r) \to \alpha \to r$$

The discrete case:

$$\mathsf{PDF}_p = \lambda x.P(p \star \lambda y.\eta(y = x))$$

The continuous case:

$$\mathsf{PDF}_p = \lambda x.\frac{d}{dx}[P(p \star \lambda y.\eta(y \leq x))]$$

RSA: implementation

- u: the type of utterances
- σ: the type of world states (sampled from a prior W)
- π: the type of linguistic parameters (sampled from a prior Θ)

RSA: implementation

- u: the type of utterances
- σ: the type of world states (sampled from a prior W)
- π: the type of linguistic parameters (sampled from a prior Θ)

$$L_1 : u \rightarrow (\sigma \times \pi \rightarrow r) \rightarrow r$$

RSA: implementation

- u: the type of utterances
- σ: the type of world states (sampled from a prior W)
- π: the type of linguistic parameters (sampled from a prior Θ)

$$L_1 : u \to (\sigma \times \pi \to r) \to r$$

$$L_1(u_0) = W \star \lambda w.\Theta \star \lambda\theta.\mathit{factor}(\mathrm{PDF}_{S_1(w,\theta)}(u_0)) \star \lambda\diamond.\eta(w,\theta)$$

RSA: implementation

- u: the type of utterances
- σ: the type of world states (sampled from a prior W)
- π: the type of linguistic parameters (sampled from a prior Θ)

$$L_1 : u \rightarrow (\sigma \times \pi \rightarrow r) \rightarrow r$$

$$L_1(u_0) = W \star \lambda w.\Theta \star \lambda\theta.\mathit{factor}(\mathrm{PDF}_{S_1(w,\theta)}(u_0)) \star \lambda\diamond.\eta(w,\theta)$$

$$S_1 : \sigma \times \pi \rightarrow (u \rightarrow r) \rightarrow r$$

RSA: implementation

- u: the type of utterances
- σ: the type of world states (sampled from a prior W)
- π: the type of linguistic parameters (sampled from a prior Θ)

$$L_1 : u \to (\sigma \times \pi \to r) \to r$$

$$L_1(u_0) = W \star \lambda w.\Theta \star \lambda\theta.\mathit{factor}(\mathrm{PDF}_{S_1(w,\theta)}(u_0)) \star \lambda\diamond.\eta(w,\theta)$$

$$S_1 : \sigma \times \pi \to (u \to r) \to r$$

$$S_1(w,\theta) = U \star \lambda u.\mathit{factor}(\mathrm{PDF}_{L_0(u,\theta)}(w) * e^{-C(u)})^{\alpha} \star \lambda\diamond.\eta(u)$$

RSA: implementation

- u: the type of utterances
- σ: the type of world states (sampled from a prior W)
- π: the type of linguistic parameters (sampled from a prior Θ)

$$\begin{aligned} L_1 &: u \rightarrow (\sigma \times \pi \rightarrow r) \rightarrow r \\ L_1(u_0) &= W \star \lambda w.\Theta \star \lambda\theta.\mathit{factor}(\mathrm{PDF}_{S_1(w,\theta)}(u_0)) \star \lambda\diamond.\eta(w,\theta) \end{aligned}$$

$$\begin{aligned} S_1 &: \sigma \times \pi \rightarrow (u \rightarrow r) \rightarrow r \\ S_1(w,\theta) &= U \star \lambda u.\mathit{factor}(\mathrm{PDF}_{L_0(u,\theta)}(w) * e^{-C(u)})^{\alpha} \star \lambda\diamond.\eta(u) \end{aligned}$$

$$L_0 : u \times \pi \rightarrow (\sigma \rightarrow r) \rightarrow r$$

RSA: implementation

- u: the type of utterances
- σ: the type of world states (sampled from a prior W)
- π: the type of linguistic parameters (sampled from a prior Θ)

$$L_1 : u \to (\sigma \times \pi \to r) \to r$$

$$L_1(u_0) = W \star \lambda w.\Theta \star \lambda\theta.\mathit{factor}(\mathrm{PDF}_{S_1(w,\theta)}(u_0)) \star \lambda\diamond.\eta(w,\theta)$$

$$S_1 : \sigma \times \pi \to (u \to r) \to r$$

$$S_1(w,\theta) = U \star \lambda u.\mathit{factor}(\mathrm{PDF}_{L_0(u,\theta)}(w) * e^{-C(u)})^{\alpha} \star \lambda\diamond.\eta(u)$$

$$L_0 : u \times \pi \to (\sigma \to r) \to r$$

$$L_0(u,\theta) = W \star \lambda w.\mathit{observe}(⦇u⦈^{\langle w,\theta\rangle}) \star \lambda\diamond.\eta(w)$$

RSA: implementation

- u: the type of utterances
- σ: the type of world states (sampled from a prior W)
- π: the type of linguistic parameters (sampled from a prior Θ)

$$L_1 : u \to (\sigma \times \pi \to r) \to r$$

$$L_1(u_0) = W \star \lambda w.\Theta \star \lambda\theta.factor(\mathrm{PDF}_{S_1(w,\theta)}(u_0)) \star \lambda\diamond.\eta(w,\theta)$$

$$S_1 : \sigma \times \pi \to (u \to r) \to r$$

$$S_1(w,\theta) = U \star \lambda u.factor(\mathrm{PDF}_{L_0(u,\theta)}(w) * e^{-C(u)})^{\alpha} \star \lambda\diamond.\eta(u)$$

$$L_0 : u \times \pi \to (\sigma \to r) \to r$$

$$L_0(u,\theta) = W \star \lambda w.observe(⦇u⦈^{\langle w,\theta\rangle}) \star \lambda\diamond.\eta(w)$$

RSA: implementation

- u: the type of utterances
- σ: the type of world states (sampled from a prior W)
- π: the type of linguistic parameters (sampled from a prior Θ)

$$L_1 : u \rightarrow (\sigma \times \pi \rightarrow r) \rightarrow r$$

$$L_1(u_0) = W \star \lambda w.\Theta \star \lambda\theta.\mathit{factor}(\mathrm{PDF}_{S_1(w,\theta)}(u_0)) \star \lambda\diamond.\eta(w, \theta)$$

$$S_1 : \sigma \times \pi \rightarrow (u \rightarrow r) \rightarrow r$$

$$S_1(w, \theta) = U \star \lambda u.\mathit{factor}(\mathrm{PDF}_{L_0(u,\theta)}(w) * e^{-C(u)})^{\alpha} \star \lambda\diamond.\eta(u)$$

$$L_0 : u \times \pi \rightarrow (\sigma \rightarrow r) \rightarrow r$$

$$L_0(u, \theta) = W \star \lambda w.\mathit{observe}(\llparenthesis u \rrparenthesis^{\langle w,\theta\rangle}) \star \lambda\diamond.\eta(w)$$

Note the different types of L_0 and L_1.

RSA: a revised version

This can be improved!

RSA: a revised version

This can be improved! Using contexts!

RSA: a revised version

This can be improved! Using contexts!

- The types of L_0 and L_1 can be made the same.

RSA: a revised version

This can be improved! Using contexts!

- The types of L_0 and L_1 can be made the same.
- Doing so allows distributions over world states and linguistic parameters to be merged into one over contexts.

RSA: a revised version

This can be improved! Using contexts!

- The types of L_0 and L_1 can be made the same.
- Doing so allows distributions over world states and linguistic parameters to be merged into one over contexts.
- Say the type of the context is some $\kappa = \alpha_1 \times ... \times \alpha_n$...

RSA: a revised version

This can be improved! Using contexts!

- The types of L_0 and L_1 can be made the same.
- Doing so allows distributions over world states and linguistic parameters to be merged into one over contexts.
- Say the type of the context is some $\kappa = \alpha_1 \times ... \times \alpha_n$…

RSA: a revised version

This can be improved! Using contexts!

- The types of L_0 and L_1 can be made the same.
- Doing so allows distributions over world states and linguistic parameters to be merged into one over contexts.
- Say the type of the context is some $\kappa = \alpha_1 \times ... \times \alpha_n$. . .

$$L_1 : u \to (\kappa \to r) \to r$$

RSA: a revised version

This can be improved! Using contexts!

- The types of L_0 and L_1 can be made the same.
- Doing so allows distributions over world states and linguistic parameters to be merged into one over contexts.
- Say the type of the context is some $\kappa = \alpha_1 \times ... \times \alpha_n$...

$$L_1 : u \to (\kappa \to r) \to r$$

$$L_1(u) = K \star \lambda\kappa.\mathit{factor}(\mathrm{PDF}_{S_1(\kappa)}(u)) \star \lambda\diamond.\eta(\kappa)$$

RSA: a revised version

This can be improved! Using contexts!

- The types of L_0 and L_1 can be made the same.
- Doing so allows distributions over world states and linguistic parameters to be merged into one over contexts.
- Say the type of the context is some $\kappa = \alpha_1 \times ... \times \alpha_n$...

$$L_1 : u \to (\kappa \to r) \to r$$

$$L_1(u) = K \star \lambda\kappa.\mathit{factor}(\mathrm{PDF}_{S_1(\kappa)}(u)) \star \lambda\diamond.\eta(\kappa)$$

$$S_1 : \kappa \to (u \to r) \to r$$

RSA: a revised version

This can be improved! Using contexts!

- The types of L_0 and L_1 can be made the same.
- Doing so allows distributions over world states and linguistic parameters to be merged into one over contexts.
- Say the type of the context is some $\kappa = \alpha_1 \times ... \times \alpha_n$...

$$L_1 : u \to (\kappa \to r) \to r$$

$$L_1(u) = K \star \lambda\kappa.\mathit{factor}(\mathrm{PDF}_{S_1(\kappa)}(u)) \star \lambda\diamond.\eta(\kappa)$$

$$S_1 : \kappa \to (u \to r) \to r$$

$$S_1(\kappa) = U^* \star \lambda u.\mathit{factor}(\mathrm{PDF}_{L_0(u)}(\kappa)^{\alpha}) \star \lambda\diamond.\eta(u)$$

RSA: a revised version

This can be improved! Using contexts!

- The types of L_0 and L_1 can be made the same.
- Doing so allows distributions over world states and linguistic parameters to be merged into one over contexts.
- Say the type of the context is some $\kappa = \alpha_1 \times \ldots \times \alpha_n$...

$$L_1 : u \to (\kappa \to r) \to r$$

$$L_1(u) = K \star \lambda\kappa.\mathit{factor}(\mathrm{PDF}_{S_1(\kappa)}(u)) \star \lambda\diamond.\eta(\kappa)$$

$$S_1 : \kappa \to (u \to r) \to r$$

$$S_1(\kappa) = U^* \star \lambda u.\mathit{factor}(\mathrm{PDF}_{L_0(u)}(\kappa)^\alpha) \star \lambda\diamond.\eta(u)$$

$$L_0 : u \to (\kappa \to r) \to r$$

RSA: a revised version

This can be improved! Using contexts!

- The types of L_0 and L_1 can be made the same.
- Doing so allows distributions over world states and linguistic parameters to be merged into one over contexts.
- Say the type of the context is some $\kappa = \alpha_1 \times ... \times \alpha_n$...

$$L_1 : u \to (\kappa \to r) \to r$$

$$L_1(u) = K \star \lambda\kappa.\mathit{factor}(\mathrm{PDF}_{S_1(\kappa)}(u)) \star \lambda\diamond.\eta(\kappa)$$

$$S_1 : \kappa \to (u \to r) \to r$$

$$S_1(\kappa) = U^* \star \lambda u.\mathit{factor}(\mathrm{PDF}_{L_0(u)}(\kappa)^{\alpha}) \star \lambda\diamond.\eta(u)$$

$$L_0 : u \to (\kappa \to r) \to r$$

$$L_0(u) = K \star \lambda\kappa.\mathit{observe}(\llparenthesis u \rrparenthesis^{\kappa}) \star \lambda\diamond.\eta(\kappa)$$

Conclusion

Summing up

It is possible to have probabilistic semantics for natural language that relies on the same machinery as used for logical interpretations:

Summing up

It is possible to have probabilistic semantics for natural language that relies on the same machinery as used for logical interpretations:

- typed λ-calculus

Summing up

It is possible to have probabilistic semantics for natural language that relies on the same machinery as used for logical interpretations:

- typed λ-calculus
- functional application

Summing up

It is possible to have probabilistic semantics for natural language that relies on the same machinery as used for logical interpretations:

- typed λ-calculus
- functional application

Summing up

It is possible to have probabilistic semantics for natural language that relies on the same machinery as used for logical interpretations:

- typed λ-calculus
- functional application

This semantics allows one to characterize:

Summing up

It is possible to have probabilistic semantics for natural language that relies on the same machinery as used for logical interpretations:

- typed λ-calculus
- functional application

This semantics allows one to characterize:

- probability distributions over possible denotations

Summing up

It is possible to have probabilistic semantics for natural language that relies on the same machinery as used for logical interpretations:

- typed λ-calculus
- functional application

This semantics allows one to characterize:

- probability distributions over possible denotations
- probabilities for formulae (given some distribution over contexts)

Summing up

It is possible to have probabilistic semantics for natural language that relies on the same machinery as used for logical interpretations:

- typed λ-calculus
- functional application

This semantics allows one to characterize:

- probability distributions over possible denotations
- probabilities for formulae (given some distribution over contexts)
- Bayesian update (or marginalization)

Summing up

It is possible to have probabilistic semantics for natural language that relies on the same machinery as used for logical interpretations:

- typed λ-calculus
- functional application

This semantics allows one to characterize:

- probability distributions over possible denotations
- probabilities for formulae (given some distribution over contexts)
- Bayesian update (or marginalization)
- semantic learning, RSA models

Summing up

It is possible to have probabilistic semantics for natural language that relies on the same machinery as used for logical interpretations:

- typed λ-calculus
- functional application

This semantics allows one to characterize:

- probability distributions over possible denotations
- probabilities for formulae (given some distribution over contexts)
- Bayesian update (or marginalization)
- semantic learning, RSA models

Summing up

It is possible to have probabilistic semantics for natural language that relies on the same machinery as used for logical interpretations:

- typed λ-calculus
- functional application

This semantics allows one to characterize:

- probability distributions over possible denotations
- probabilities for formulae (given some distribution over contexts)
- Bayesian update (or marginalization)
- semantic learning, RSA models

…using the same logical language one uses to characterize linguistic meanings.

References

Goodman, Noah D., and Michael C. Frank. 2016. Pragmatic Language Interpretation as Probabilistic Inference. *Trends in Cognitive Sciences* 20:818–829.

Goodman, Noah D., Vikash K. Mansinghka, Daniel Roy, Keith Bonawitz, and Joshua B. Tenenbaum. 2008. Church: a language for generative models. In *Proceedings of the Twenty-Fourth Conference on Uncertainty in Artificial Intelligence*, UAI'08, 220–229. Arlington, Virginia, USA: AUAI Press.

Goodman, Noah D., and Andreas Stuhlmüller. 2013. Knowledge and Implicature: Modeling Language Understanding as Social Cognition. *Topics in Cognitive Science* 5:173–184.

Lassiter, Daniel, and Noah D. Goodman. 2013. Context, scale structure, and statistics in the interpretation of positive-form adjectives. *Semantics and Linguistic Theory* 23:587–610. Number: 0.

Lassiter, Daniel, and Noah D. Goodman. 2017. Adjectival vagueness in a Bayesian model of interpretation. *Synthese* 194:3801–3836.