Probabilistic compositional semantics, purely

Julian Grove and Jean-Philippe Bernardy

LENLS18, November 14, 2021

CLASP, University of Gothenburg

Motivation

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).

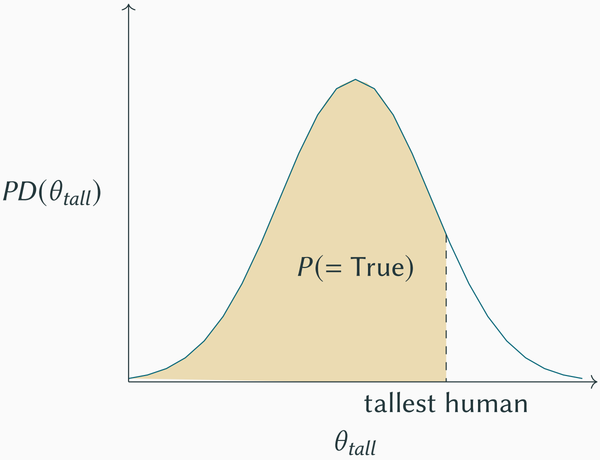
Such programs *describe* probability distributions over logical meanings.

Schematically...

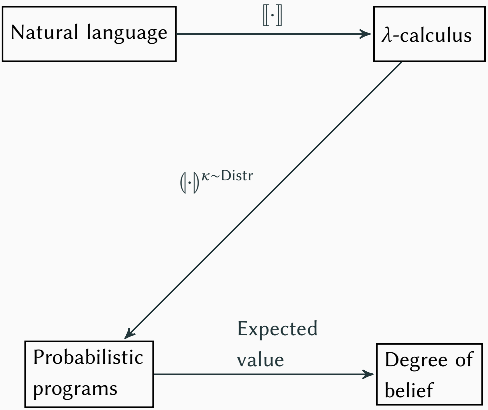
$$[\![\textit{someone is tall}]\!] \; = \; \exists x : \text{human}(x) \land \text{height}(x) \geq \theta_{tall}$$

Schematically...

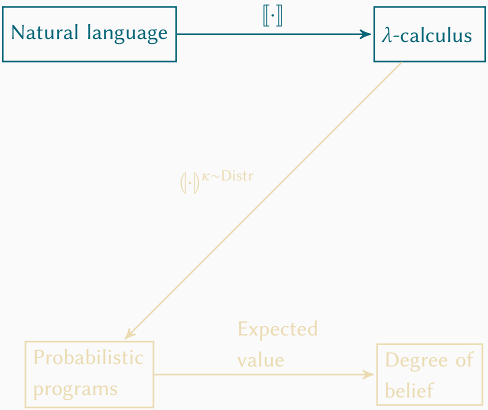
$$[\![\textit{someone is tall}]\!] = \exists x : \text{human}(x) \wedge \text{height}(x) \geq \theta_{tall}$$



Our system


Natural language
⟦·⟧
λ-calculus
⦇·⦈κ∼Distr
Probabilistic programs
Expected value
Degree of belief

Up next



Natural language
$[\![\cdot]\!]$
λ-calculus
$(\!|\cdot|\!)^{\kappa\sim\mathrm{Distr}}$
Probabilistic programs
Expected value
Degree of belief

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.

$$\llbracket \textit{someone} \rrbracket = \lambda k.\exists x : \mathrm{human}(x) \wedge k(x)$$
$$\llbracket \textit{is} \rrbracket = \lambda x.x$$

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.

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

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

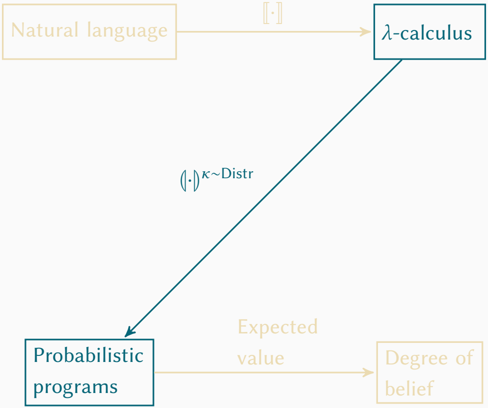
(1) Someone is tall.

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

Functional application and β-reduction:

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

Up next


Natural language
⟦·⟧
λ-calculus
⦇·⦈$^{\kappa\sim\mathrm{Distr}}$
Probabilistic programs
Expected value
Degree of belief

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) $\theta_{tall} : d_{tall}$

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

A λ-homomorphism in a context

Given some context κ:

$$(\!|c_i|\!)^{\kappa} = \kappa_i \qquad\qquad (c_i \text{ is the } i^{th} \text{ constant})$$

A λ-homomorphism in a context

Given some context κ:

$$(\!|c_i|\!)^\kappa = \kappa_i \qquad (c_i \text{ is the } i^{th} \text{ constant})$$

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

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

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

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

$$(\!|M_i|\!)^\kappa = (\!|M|\!)^\kappa_i \qquad \text{(projection)}$$

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:

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

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:

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

- $(\!|[\![\textit{someone is tall}]\!]|\!)^\kappa = \exists x : \kappa_2(x) \wedge \kappa_3(\kappa_1(x))(\kappa_4)$
- 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 $(\alpha \to r) \to r$ *returns* values of type α.

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

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.

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.
    - $\mathcal{N}(\mu, \sigma)(f) = \int_{-\infty}^{\infty} \mathrm{PDF}_{\mathcal{N}(\mu,\sigma)}(x) * f(x) dx$

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:

$$\begin{aligned} \eta &: \alpha \to (\alpha \to r) \to r && \text{('return')} \\ \eta(a) &= \lambda f.f(a) \end{aligned}$$

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:

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

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.

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

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.

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

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

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

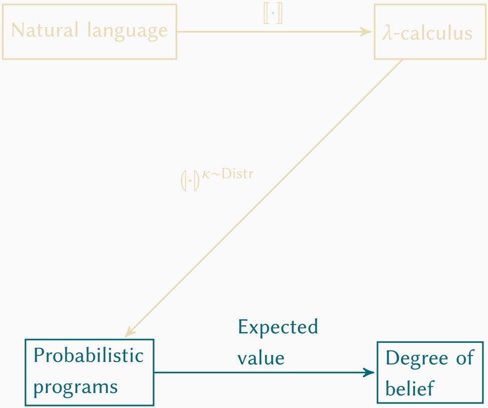
Building probabilistic programs

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 $(\alpha_1 \times ... \times \alpha_n \rightarrow r) \rightarrow r$.
- Then, for a sentence ϕ in the logical language, we may do:

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

Up next


Natural language
⟦·⟧
λ-calculus
⦇·⦈κ~Distr
Probabilistic programs
Expected value
Degree of belief

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

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

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. $\theta_{tall} : d_{tall}$

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)

$$
\begin{aligned}
& K \star \lambda\kappa.\eta(\llparenthesis \exists x : \mathrm{human}(x) \wedge \mathrm{height}(x) \geq \theta_{tall} \rrparenthesis^{\kappa}) \\
& \vdots \\
= \; & \lambda f.\, \mathcal{N}(72, 3)(\lambda d. f(\exists x : human(x) \wedge height(x) \geq d)) \\
= \; & \lambda f.\, \int_{-\infty}^{\infty} \mathrm{PDF}_{\mathcal{N}(72,3)}(y) * f(\exists x : human(x) \wedge height(x) \geq y) dy
\end{aligned}
$$

An example (cont'd) (cont'd)

Computing a probability:

An example (cont'd) (cont'd)

Computing a probability:

$$\frac{\int_{-\infty}^{\infty} \mathsf{PDF}_{\mathcal{N}(72,3)}(y) * \mathbb{1}(\exists x : \mathit{human}(x) \wedge \mathit{height}(x) \geq y) dy}{\int_{-\infty}^{\infty} \mathsf{PDF}_{\mathcal{N}(72,3)}(y) dy}$$

An example (cont'd) (cont'd)

Computing a probability:

$$\frac{\int_{-\infty}^{\infty} \mathrm{PDF}_{\mathcal{N}(72,3)}(y) * \mathbb{1}(\exists x : \mathit{human}(x) \land \mathit{height}(x) \geq y)dy}{\int_{-\infty}^{\infty} \mathrm{PDF}_{\mathcal{N}(72,3)}(y)dy}$$

$$= \int_{-\infty}^{\infty} \mathrm{PDF}_{\mathcal{N}(72,3)}(y) * \mathbb{1}(\exists x : \mathit{human}(x) \land \mathit{height}(x) \geq y)dy$$

An example (cont'd) (cont'd)

Computing a probability:

$$\frac{\int_{-\infty}^{\infty} \mathsf{PDF}_{\mathcal{N}(72,3)}(y) * \mathbb{1}(\exists x : \mathit{human}(x) \wedge \mathit{height}(x) \geq y) dy}{\int_{-\infty}^{\infty} \mathsf{PDF}_{\mathcal{N}(72,3)}(y) dy}$$

$$= \int_{-\infty}^{\infty} \mathsf{PDF}_{\mathcal{N}(72,3)}(y) * \mathbb{1}(\exists x : \mathit{human}(x) \wedge \mathit{height}(x) \geq y) dy$$

Computing a probability:

$$\frac{\int_{-\infty}^{\infty} \mathrm{PDF}_{\mathcal{N}(72,3)}(y) * \mathbb{1}(\exists x : \mathit{human}(x) \land \mathit{height}(x) \geq y)dy}{\int_{-\infty}^{\infty} \mathrm{PDF}_{\mathcal{N}(72,3)}(y)dy}$$

$$= \int_{-\infty}^{\infty} \mathrm{PDF}_{\mathcal{N}(72,3)}(y) * \mathbb{1}(\exists x : \mathit{human}(x) \land \mathit{height}(x) \geq y)dy$$

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

Bayesian inference (e.g., RSA)

RSA models: a popular application of probabilistic semantics.

RSA models: a popular application of probabilistic semantics.

The basic idea:

RSA

RSA models: a popular application of probabilistic semantics.

The basic idea:

- The RSA framework models a pragmatic listener, L_1…

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.

Factoring by a weight / observing a premise

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

Factoring by a weight / observing a premise

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

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

RSA: implementation

Say the type of the context is $\kappa = \alpha_1 \times ... \times \alpha_n$…

RSA: implementation

Say the type of the context is $\kappa = \alpha_1 \times ... \times \alpha_n$…

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

RSA: implementation

Say the type of the context is $\kappa = \alpha_1 \times ... \times \alpha_n$…

$$L_0 : u \to (\kappa \to r) \to r$$
$$L_0(u) = K \star \lambda\kappa.observe(⦇u⦈^{\kappa}) \star \lambda\diamond.\eta(\kappa)$$

RSA: implementation

Say the type of the context is $\kappa = \alpha_1 \times ... \times \alpha_n$…

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

$$L_0(u) = K \star \lambda\kappa.observe(⦇u⦈^\kappa) \star \lambda\diamond.\eta(\kappa)$$

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

RSA: implementation

Say the type of the context is $\kappa = \alpha_1 \times ... \times \alpha_n$...

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

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

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

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

RSA: implementation

Say the type of the context is $\kappa = \alpha_1 \times ... \times \alpha_n$…

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

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

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

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

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

RSA: implementation

Say the type of the context is $\kappa = \alpha_1 \times ... \times \alpha_n$...

$$L_0 : u \to (\kappa \to r) \to r$$
$$L_0(u) = K \star \lambda\kappa.observe((\!|u|\!)^\kappa) \star \lambda\diamond.\eta(\kappa)$$

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

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

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)
- RSA models (also, semantic learning)

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)
- RSA models (also, semantic learning)

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)
- RSA models (also, semantic learning)

…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.