Probabilistic compositional semantics, purely

Julian Grove and Jean-Philippe Bernardy

Centre for Linguistic Theory and Studies in Probability
Department of Philosophy, Linguistics and Theory of Science
University of Gothenburg
`firstname.lastname@gu.se`

Abstract. We provide a general framework for the integration of formal semantics with probabilistic reasoning. This framework is conservative, in the sense that it relies only on typed λ-calculus and is thus compatible with logical systems already in use. The framework is also presented modularly, in that it regards probabilistic effects (i.e., sampling and marginalization) as *side effects*, using continuations. We show how our framework may be used to build probabilistic programs compositionally within typed λ-calculus and then illustrate its use on two applications: semantic learning and pragmatic inference within the Rational Speech Act framework.

1 Introduction

Formal semantics in the tradition of Montague characterizes linguistic meaning in terms of either a logic or a model, constructed set-theoretically. By exploiting an already well understood formalism, a logical characterization of meaning allows one to reason about it in terms of notions like entailment. Indeed, while the formal description such a characterization provides is necessarily abstract, it can be assembled compositionally, in terms of rules that combine the meanings of syntactic constituents. It is this feature of formal semantics that makes it such an attractive approach to meaning, and one which has persisted throughout the development of the field.

There has been much effort in the last decade to connect formal semantics to mathematically explicit models of pragmatic reasoning, with Rational Speech Act (RSA) models providing a paradigmatic case. RSA models consider utterance interpretation to be a process of updating probability distributions over logically characterized meanings [7, 10, 14, 15]. In doing so, they aim to capture a central feature of discourse known since the work of Grice [11]; namely, that it is constrained by principles of appropriate social behavior, which, through the reasoning of interlocutors, serve to enrich the very meanings which are communicated.

The present work provides a general approach to the integration of formal semantics with probabilistic reasoning — one which is both modular and conservative. Past efforts to consider linguistic meaning probabilistically (including the formative work of Goodman and Lassiter) have drastically modified the underlying logic to express it, typically in a way that freely mixes a logical semantics with probabilities. Goodman and Lassiter [8], for example, encode meanings using the probabilistic programming language Church [9], a decision which constitutes a radical departure from formal semantics in the style of Montague: while the latter uses a pure λ-calculus, Church programs can invoke probabilistic effects, i.e., by sampling from or updating a distribution at any point in a given program.

In contrast to this and similar approaches to probabilistic semantics, ours allows for the usual approach to compositional semantics, in terms of a *pure* logical language. Moreover, we expect that our approach will be quite general: it in principle allows for any simply typed language with products, but it should also be compatible with more expressive systems, e.g., System F [6] and dependent type theory (we defer an investigation of the generality of our approach, however). Our trick is to treat probabilistic computation modularly, as a side effect, using continuations. Doing so allows logical meanings to be viewed as values computed by probabilistic programs. Even so, as we shall see, our semantics does not overstep the tight bounds of typed λ-calculus: probabilistic programs are *themselves* expressed using the same logic. We can thus provide an expressive probabilistic compositional semantics without the use of radically novel tools.

2 Formal semantics

To illustrate, we provide a schematic English fragment, which we translate into a higher-order language with types for individuals (e), truth values (t), and real numbers (r). In addition to function types ($\alpha \to \beta$), we assume access to products ($\alpha \times \beta$ and unit type $\diamond$), along with their associated constructors $\langle M, N\rangle : \alpha \times \beta$ (for $M : \alpha$ and $N : \beta$), destructors $\pi_1 M : \alpha$ and $\pi_2 M : \beta$ (for $M : \alpha \times \beta$), and unit $\diamond : \diamond$, as well as n-ary generalizations of these. Notably, we employ an indicator function $\mathbb{1} : t \to r$ taking $\top$ ('true') and $\bot$ ('false') onto 1 and 0, respectively. We additionally assume the existence of a family d_i of subtypes of r corresponding to degree types. For instance d_{tall} represents degrees of height, d_{happy} degrees of happiness, etc.

In general, we assume the language to have, among the non-logical constants, a finite number to be assigned probabilistic interpretations. We call such constants "special" constants. For our example, we employ the following:

$$\mathsf{person} : e \to t \qquad \mathsf{height} : e \to d_{tall} \qquad \theta_{tall} : d_{tall} \qquad (\geq) : r \to r \to t$$

In terms of these, the following meanings can be given for *someone*, *is*, and *tall*, in order to derive the meaning of *someone is tall* via functional application:

$$\begin{aligned} [\![someone]\!] &= \lambda k.\exists x : \mathsf{person}(x) \land k(x) \\ [\![is]\!] &= \lambda x.x \\ [\![tall]\!] &= \lambda x.\mathsf{height}(x) \geq \theta_{tall} \end{aligned}$$

The meaning of *someone is tall*, $[\![someone]\!]([\![is]\!]([\![tall]\!]))$, can then be computed to be $\exists x : \mathsf{person}(x) \land \mathsf{height}(x) \geq \theta_{tall}$.

3 The traditional interpretation

For completeness, we spell out the "traditional" interpretation of the logical language illustrated above, in terms of an interpretation function, $(\!|\cdot|\!)$, which is given by a λ-homomorphism:[1]

$$
\begin{aligned}
&(\!|x|\!) = x && (x \text{ is a variable})\\
&(\!|\lambda x.M|\!) = \lambda x.(\!|M|\!)\\
&(\!|M(N)|\!) = (\!|M|\!)((\!|N|\!))\\
&(\!|\langle M, N\rangle|\!) = \langle (\!|M|\!), (\!|N|\!)\rangle\\
&(\!|\pi_i M|\!) = \pi_i (\!|M|\!)\\
&(\!|\theta_{tall}|\!) = d\\
&(\!|\mathsf{height}|\!) = height\\
&(\!|\mathsf{person}|\!) = person\\
&(\!|(\geq)|\!) = (\geq)
\end{aligned}
$$

Here, $d : d_{tall}$ is some real number representing the contextual standard of height used by the adjective *tall*, $height : e \rightarrow d_{tall}$ is some function from individuals to real numbers, and $person : e \rightarrow t$ is some function from individuals to truth values. The constant $(\geq) : r \rightarrow r \rightarrow r$ is intended to be the "greater-than-or-equal-to" relation on real numbers. To save space, we have left implicit the interpretation of the other constants, such as $\top$, $\bot$, and $\exists$, which is standard. One can now compose $(\!|\cdot|\!)$ with $[\![\cdot]\!]$ and map *someone is tall* onto $\exists x : person(x) \wedge height(x) \geq d$, i.e., a (formula representing a) truth value.

4 The probabilistic interpretation

We provide the probabilistic interpretation in two steps. First, we parameterize our interpretation function, $(\!|\cdot|\!)^{\kappa}$, by a tuple κ of values, which we call a *context*. The idea is to use such a tuple to provide an interpretation for the special constants. In particular, we assume that the n special constants of the language are ordered, such that, if constant c_i has type α_i, then $\kappa : \alpha_1 \times ... \times \alpha_n$. For any

[1] The λ-homomorphisms that we employ map one higher-order language into another, preserving variables, abstractions, applications, pairing, and projection. They are accompanied by type-homomorphisms $\overline{\alpha}$ which, for us, preserve implication and products (i.e., $\overline{\alpha \rightarrow \beta} = \overline{\alpha} \rightarrow \overline{\beta}$ and $\overline{\alpha \times \beta} = \overline{\alpha} \times \overline{\beta}$), but which may in principle affect base types. In general, if $M : \alpha$, then $(\!|M|\!) : \overline{\alpha}$. The motivation for these constraints is that they provide meanings to the constants of the source language, leaving the surrounding λ-calculus unaffected (as analogous to a traditional model-theoretic interpretation). In this case, both $(\!|\cdot|\!)$ and its associated type homomorphism are trivial, mapping both constants and base types onto themselves.

such κ, $(\!|\cdot|\!)^\kappa$ is the following λ-homomorphism:

$$
\begin{aligned}
(\!|x|\!)^\kappa &= x && (x \text{ is a variable})\\
(\!|\lambda x.M|\!)^\kappa &= \lambda x.(\!|M|\!)^\kappa\\
(\!|M(N)|\!)^\kappa &= (\!|M|\!)^\kappa((\!|N|\!)^\kappa)\\
(\!|\langle M, N\rangle|\!)^\kappa &= \langle (\!|M|\!)^\kappa, (\!|N|\!)^\kappa\rangle\\
(\!|\pi_i M|\!)^\kappa &= \pi_i (\!|M|\!)^\kappa\\
(\!|c_i|\!)^\kappa &= \pi_i\kappa && (c_i \text{ is the } i^{th} \text{ special constant})
\end{aligned}
$$

Thus if c_i is one of θ_{tall}, height, person, or $\geq$, then its interpretation is determined by the context κ. (Again, we have omitted the interpretation of other constants to save space.) Obviously, if c_i is of type α_i, then so is $\pi_i\kappa$. We assume that all probabilistic semantic knowledge resides in the interpretation of special constants, and thus ultimately in the context κ. It remains to be shown how to evaluate the above expressions when κ is a random variable.

5 Probabilistic programs

In general, we consider something a random variable if it is the value returned by a probabilistic program. In our framework, a probabilistic program returning values of type α is a function of type $(\alpha \to r) \to r$; that is, one which consumes a *projection* function (from values of type α to real numbers), in order to return a real number.[2] The intent is that if p is a probabilistic program and f is a projection function, then $p(f)$ is the sum of $f(x)$, for all possible values of x returned by the program, weighted in proportion to their probabilities.

Given this setup, probabilistic programs form a *monad*. A monad, as stated in Figure 1, is a functor M from types to types, associated with two operators, η ('return') and $\star$ ('bind'), satisfying certain laws. In general, implementing a monad in a pure setting, such as the λ-calculus, allows one to simulate various notions of side effect, including probabilistic computation, as we shall see. The role of η is to inject an ordinary value into the monad, while that of $\star$ is to compose *computations*. More precisely, $\star$ runs a computation of type $M\alpha$, and then binds the returned value to a variable in the next computation (something of type $\alpha \to M\beta$). In the case of probabilistic programs, $M\alpha = (\alpha \to r) \to r$,

[2] There is some precedent for this representation of probabilistic programs, by Mohammed Ismail and Shan [17], who describe a small typed probabilistic programming language and provide a denotational semantics for it in terms of continuations. Our formulation is chiefly inspired by the dependently typed language of Bernardy et al. [3]. See also Jansson et al. [13].

Operators

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

Laws on terms

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

Fig. 1. Definition of a monad

and the return η and bind operator $\star$ are inherited from the continuation monad:

$$\begin{aligned}\eta &: \alpha \to (\alpha \to r) \to r\\ \eta(a) &= \lambda c.c(a)\end{aligned}$$

$$\begin{aligned}(\star) &: ((\alpha \to r) \to r) \to\\ &\quad (\alpha \to (\beta \to r) \to r) \to\\ &\quad (\beta \to r) \to r\\ m \star k &= \lambda c.m(\lambda x.k(x)(c))\end{aligned}$$

By employing the monadic operators, one may sequence a probabilistic program $p : (\alpha \to r) \to r$ with some projection function $f : \alpha \to r$ by binding the value returned by p to a variable x and returning $f(x)$ (via η):

$$p \star \lambda x.\eta(f(x)) : (r \to r) \to r$$

Indeed, feeding the identity function of type $r \to r$ to the result obtains $p(f) : r$.

The encoding of probabilistic programs in terms of continuations may at first appear somewhat opaque and indirect. In general, one can see a continuation (here, of type $\alpha \to r$) as a question to ask a program. The result type r restricts the sorts of questions one may ask, i.e., to those having real numbers as answers. As if responding with a riddle, moreover, the probabilistic program returns the weighted sum of the answers for its possible values. This means, for instance, that given a probabilistic program p, one may feed it the question $(\lambda x.1)$, which asks how much mass it assigns to any given value x, in order to get the answer $p(\lambda x.1)$, which is just the total mass assigned by p.

If p returns truth values (i.e., if it is of type $(t \to r) \to r$), we can ask for the mass it assigns to $\top$ by passing the indicator function as a continuation: $p(\mathbb{1})$. Consequently, we may compute a *probability* for p as the expected value of $\mathbb{1}$, in terms of a function $P : ((t \to r) \to r) \to r$:

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

The denominator (the total mass assigned by p) normalizes the result.

Now, let K be a probabilistic program representing the distribution of contexts; that is, if $\alpha_1 \times ... \times \alpha_n$ is the type of contexts, K is of type $(\alpha_1 \times ... \times \alpha_n \to r) \to r$. Given a term ϕ of type t, we encode its interpretation in the context of K as the following probabilistic program:

$$K \star \lambda\kappa.\eta(\llparenthesis\phi\rrparenthesis^{\kappa})$$

Like all probabilistic programs returning truth values, the above is of type $(t \to r) \to r$. Operationally, it reads in the random context returned by K and computes from it a truth value for ϕ in this context. As such, one can determine a probability for it, as outlined above.

To illustrate, consider our running example, *someone is tall*, to which we assigned the interpretation $\exists x : \mathsf{person}(x) \land \mathsf{height}(x) \geq \theta_{tall}$. Let us assume a probabilistic program K returning contexts where the interpretation of θ_{tall} is a random variable having a normal distribution with a mean of 72 inches and a standard deviation of 3 inches. Moreover, we assume that the interpretations of the other special constants are fixed as the functions *height*, *person*, and $\geq$, as above. Then, assuming that the order of the constants is height, person, $\geq$, θ_{tall}, we have the following definition of K:

$$\begin{aligned} &K : (((e \to d_{tall}) \times (e \to t) \times (r \to r \to t) \times d_{tall}) \to r) \to r \\ &K = \mathcal{N}(72, 3) \star \lambda d.\eta(height, person, (\geq), d) \end{aligned}$$

Here, $\mathcal{N}(72, 3)$ is a probabilistic program (of type $(d_{tall} \to r) \to r$) representing a normal distribution with the relevant mean and standard deviation. If fed a projection function f of type $d_{tall} \to r$, this program results in a real number which is gotten by integrating f over the real line, weighting each $f(d)$ by the probability of d.[3]

[3] Here, we leave $\mathcal{N} : d_{tall} \times d_{tall} \to (d_{tall} \to r) \to r$ unanalyzed. In general, computing a continuous distribution $\mathcal{D} : p_1 \times ... \times p_n \to (d \to r) \to r$ over d amounts to computing

$$\lambda\langle p_1, ..., p_n\rangle, f. \int_{-\infty}^{\infty} \mathrm{PDF}_{\mathcal{D}(p_1,...,p_n)}(x) * f(x)dx$$

where $\mathrm{PDF}_{\mathcal{D}(p_1,...,p_n)}$ provides the probability density function associated with $\mathcal{D}$ (given parameters $p_1, ..., p_n$). Such integrals don't in general admit closed-form solutions, and so one must resort to approximations. We implement this via Markov chain Monte Carlo sampling in our Haskell implementation, using the library at https://github.com/jyp/ProbProg.

Our strategy allows us to associate a probabilistic program with the sentence *someone is tall*, as follows:

$$
\begin{aligned}
& K \star \lambda\kappa.\eta(\llparenthesis \exists x : \mathsf{person}(x) \wedge \mathsf{height}(x) \geq \theta_{tall} \rrparenthesis^{\kappa}) \\
= \; & K \star \lambda\kappa.\eta(\exists x : \llparenthesis \mathsf{person} \rrparenthesis^{\kappa}(x) \wedge \llparenthesis (\geq) \rrparenthesis^{\kappa}(\llparenthesis \mathsf{height} \rrparenthesis^{\kappa}(x))(\llparenthesis \theta_{tall} \rrparenthesis^{\kappa})) \\
= \; & \mathcal{N}(72,3) \star \lambda d.(\eta(height, person, (\geq), d) && \text{(by Assoc.)} \\
& \qquad \star \lambda\kappa.\eta(\exists x : (\pi_2\kappa)(x) \wedge (\pi_3\kappa)((\pi_1\kappa)(x))(\pi_4\kappa))) \\
= \; & \mathcal{N}(72,3) \star \lambda d.\eta(\exists x : person(x) \wedge height(x) \geq d) && \text{(by Left Id.)} \\
= \; & \lambda c.\mathcal{N}(72,3)(\lambda d.c(\exists x : person(x) \wedge height(x) \geq d))
\end{aligned}
$$

As expected, we have a program of type $(t \to r) \to r$. We may therefore compute a probability for it as:

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

Because $\mathcal{N}$ represents a genuine probability distribution, its total mass is 1, and we can simply ignore the denominator:

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

The value of this expression is determined by computing the truth (i.e., either 1 or 0) of the proposition that someone's height exceeds the height threshold d at every possible value of d, and weighting it by d's probability. This model of the uncertainty associated with *someone is tall* locates it in the meaning of *tall*; in particular, how tall one must be, in order to be considered tall.

For example, consider a case in which someone is 72 inches tall and no one is taller. Then the condition imposed by the meaning of *someone is tall* will be met by all $d \leq 72$, and the sentence will be assigned the probability 0.5. In general, the probability assigned will be equal to the mass of $\mathcal{N}(72,3)$ that is less than or equal to the height of the tallest person.

6 Bayesian inference

One of the main interests of a probabilistic semantics such as the one we have proposed is that it can be used to characterize Bayesian update. For this purpose, we define the following function *observe*:

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

Given a proposition ϕ, *observe* either keeps or throws out its continuation, depending on whether ϕ is true or false; hence, the resulting program retains only values from the part of the distribution it represents compatible with ϕ being true.[4] This function thus allows us to condition the probability of ϕ on a premise ψ as follows, exploiting the monadic structure of probabilistic programs:

$$K \star \lambda\kappa.observe((\!|\psi|\!)^{\kappa}) \star \lambda\diamond.\eta((\!|\phi|\!)^{\kappa})$$

Such a conditioning process can be used for several purposes: for probabilistic inference (as suggested by our running example), but also to refine the probability distributions associated with constants; that is, for semantic learning. We briefly suggest how each of these tasks can be accomplished in our framework, starting with semantic learning.

6.1 Semantic learning

Semantic learning in our framework is matter of updating (distributions of) contexts. Given a program K_0 returning contexts which represents the initial state of one's semantic knowledge, one may observe a number of propositions to be true or false, thus obtaining a new program, K_1:

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

The effect of sequencing K_0 with such a series of observations is to zero out the portion of its distribution in which $\phi_1, ..., \phi_n$ are false, returning the values that survive.

Let us say that a learner is attempting to learn the meaning of *tall*, and they start out with a distribution of contexts such that the height threshold that the adjective makes use of ranges over a normal distribution with a mean of 68 inches and a standard deviation of 3 inches (we will deal here with the constants height, $\geq$, and θ_{tall}, along with the four names for individuals c, m, a, and v):

$$K_0 : ((e \times e \times e \times e \times (e \to d_{tall}) \times (r \to r \to t) \times d_{tall}) \to r) \to r$$
$$K_0 = \mathcal{N}(68, 3) \star \lambda d.\eta(c, m, a, v, height, (\geq), d)$$

In addition, this learner happens to know the following three facts: that Camilla is 65 inches tall, that Matt is 67 inches tall, and that Anna is 72 inches tall:

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

One day, someone this learner trusts utters the following three sentences, in sequence: (1) *Camilla isn't tall*, (2) *Matt isn't tall*, (3) *Anna is tall*. Upon hearing

[4] Some may recognize it as akin to the *guard* function of Haskell's MonadPlus and Alternative classes.

these utterances, the learner updates K_0, in order to obtain K_1:

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

This may in turn be simplified to:

$$\mathcal{N}(68, 3) \star \lambda d.observe(72 \geq d \wedge d > 67) \star \lambda\diamond.\eta(c, m, a, v, height, (\geq), d)$$

Thus K_1 is just like K_0, but for the fact that the distribution associated with θ_{tall} has been pared down to only include the mass of $\mathcal{N}(68, 3)$ in the interval $(67, 72]$. If Vlad is 68 inches tall ($height(v) = 68$), then the sentence *Vlad is tall* would have been associated with the probability 0.5 in K_0, while it is associated with a probability of around 0.24 in K_1:

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

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

6.2 RSA: background

In the case of probabilistic inference, our framework can serve as the basis for complex pragmatic reasoning, as in RSA models. For example, Lassiter and Goodman [14] present an RSA model of the inference made when someone utters a sentence such as *Vlad is tall*. This model consists of a pragmatic listener (L_1), who reasons about probable meanings based on the expected behavior of a pragmatic speaker (S_1), who, in turn, reasons about a literal listener (L_0). These agents' behaviors are modeled in terms of the following equations (adapted to the current example):

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

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

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

Each of these statements defines a probability distribution for the random variables of interest. L_1, in particular, infers a joint probability distribution for h and d_{tall}, the values of the random variables representing Vlad's height and the height threshold for the adjective *tall*, respectively. The function C in the S_1 model is utterance cost. The parameter α is the "temperature" of the S_1 model: it controls the extent to which the speaker behaves rationally, i.e., by taking the expected behavior of the literal listener L_0, as well as utterance cost, into account in designing their distribution over utterances.

Given the more general notions of a world state w and a parameter θ, (h and d_{tall}, respectively, in the example above), these equations may be presented more perspicuously as follows, given some utterance u_0:

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

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

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

Thus abstractly, pragmatic listeners provide a joint posterior distribution over world states w and parameters θ, given an utterance u_0.[5] Pragmatic speakers provide a distribution of utterances, given the particular world state w (and parameter θ) they wish to communicate. These utterances, moreover, are taken from an antecedently chosen set U of possible utterances, which is generally assumed to be finite, thus justifying the use of summation in the normalizing factor for S_1. Finally, linguistic uncertainty is represented by the parameter θ, which is passed from the pragmatic listener L_1 down to the literal listener L_0, through the speaker model S_1. Note, therefore, that L_1 differs from L_0 in a crucial respect: while L_1 samples both world states and parameters, L_0 samples only world states, relying on a parameter which has been fixed by S_1 (and L_1, in turn).

6.3 RSA: implementation

Our purpose is to illustrate how the RSA framework may be realized in the vocabulary of probabilistic programs. Taking u to be the type of utterances, s the type of world states, and θ the type of linguistic parameters, we aim to find a program L_1 of type $u \to (s \times \theta \to r) \to r$, which, given an utterance, provides a joint distribution over world states and parameters, and which satisfies the desiderata laid out above. In order to do so, it is useful to introduce the following

[5] Note that we define this posterior in terms of a joint prior distribution $P_{L_1}(w, \theta)$. Lassiter and Goodman [14] assume the prior distributions over world states and linguistic parameters to be independent, with an effectively uniform prior over parameters.

generalization of *observe* to fuzzy conditions:

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

Instead of a truth value, *factor* takes a real number and applies it as a weight to the result of its continuation. Thus *observe* may be viewed as the specific case of *factor* in which the relevant weight is either 1 or 0.[6]

Now, we may formulate L_1 as follows. Say that S_1 provides a probabilistic program returning *utterances*, given a world state and a parameter; i.e., it is of type $s \times \theta \to (u \to r) \to r$. Then given some w and θ, we would like access to the probability mass function corresponding to $S_1(w, \theta)$ — $\mathrm{PMF}_{S_1(w,\theta)}$ — of type $u \to r$, so that we may appropriately factor the probability of $\langle w, \theta \rangle$ in L_1, given an utterance. (We will come back to how we obtain the PMFs of probabilistic programs shortly. For now, we simply take them for granted.) Moreover, let us assume that world states and parameters take prior distributions $W : (s \to r) \to r$ and $\Theta : (\theta \to r) \to r$, respectively. These assumptions leave us with the following definition of L_1:

$$L_1 : u \to (s \times \theta \to r) \to r$$
$$L_1(u_0) = W \star \lambda w.\Theta \star \lambda\theta.factor(\mathrm{PMF}_{S_1(w,\theta)}(u_0)) \star \lambda\diamond.\eta(w, \theta)$$

Now, given some prior distribution U over utterances (i.e., of type $(u \to r) \to r$), we may similarly provide definitions of S_1 and L_0, where PDF_p is the probability density function associated with p, i.e., when the value p returns is continuous:

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

$$L_0 : u \times \theta \to (s \to r) \to r$$
$$L_0(u, \theta) = W \star \lambda w.observe(\llbracket u \rrbracket^{\langle w,\theta\rangle}) \star \lambda\diamond.\eta(w)$$

Note our use of notation in the definition of L_0. Here, the pair $\langle w, \theta \rangle$ provides a context in terms of which we can interpret the utterance u, which we assume is translated, via $\llbracket\cdot\rrbracket^{\langle w,\theta\rangle}$, into a formula of type t. Moreover, such a formula may be obtained by first providing a traditional Montague-style semantics to obtain a meaning of type t, and then applying the λ-homomorphism $(\!|\cdot|\!)^{\langle w,\theta\rangle}$, which replaces any special constants with w or θ, as appropriate.

Having stated our formulation of RSA somewhat abstractly, let us now turn to the problem of PMFs (and PDFs); that is, of obtaining a function of type $\alpha \to r$ from a probabilistic program of type $(\alpha \to r) \to r$. If α is discrete, we may construct its PMF as follows (recall that P takes a probabilistic program of type $(t \to r) \to r$ onto a probability):

$$\mathrm{PMF}_{(\cdot)} : ((\alpha \to r) \to r) \to \alpha \to r$$
$$\mathrm{PMF}_p = \lambda x.P(p \star \lambda y.\eta(y = x))$$

[6] That is, $observe(\phi)(f) = factor(\mathbb{1}(\phi))(f)$.

That is, for every $x : \alpha$, $\mathrm{PMF}_p(x)$ evaluates the probability that p returns x.

If α is continuous, however, we have a problem: the probability that any two values x and y are equal is zero, and the above definition (but for a PDF) would have it return zero everywhere! Fortunately, there are sound remedies which we may adopt for the continuous case. For instance, we may take the derivative of the cumulative mass of a given distribution p with respect to the argument:

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

Indeed, these two definitions may be plugged into the descriptions of L_1, S_1, and L_0 above, in order to provide them with fuller specifications.[7] One need only determine what the distributions U, W, and Θ are. To realize the model of Lassiter and Goodman [14], we would take U to be a small finite set, W to be a normal distribution, and Θ to be, effectively, uniform.[8] The resulting probabilistic program can be computed approximately using Monte Carlo methods; in this case, one will typically evaluate a probabilistic program to an approximate, finite PDF.

We close out this section by observing a noteworthy feature of the foregoing formulation of RSA: it highlights an odd lack of symmetry between the L_1 model and the L_0 model. Why does L_1 sample both world states from W and linguistic parameters from Θ, while L_0 samples only the former? Indeed, this fact is now reflected in their types! L_1 is of type $u \to (s \times \theta \to r) \to r$: it takes an utterance and returns a distribution over pairs of world states and parameters. Meanwhile, L_0 is of type $u \times \theta \to (s \to r) \to r$: it takes a pair of an utterance *and* a parameter and returns a distribution over world states. Thus L_0 considers a linguistic parameter which has been *fixed* by L_1 and S_1. Put differently, S_1 reasons about an L_0 that knows θ ahead of time, when determining what to say. Yet more vividly, the pragmatic listener assumes that the speaker is under the impression that the two have already (telepathically, perhaps) coordinated on linguistic parameters.

Maybe, it is more realistic not to assume that S_1 imagines such an omniscient L_0. In fact, relaxing this assumption restores the symmetry of the model. At the same time, it conveniently allows us not to explicitly split the context κ into two parts w and θ. As in previous sections, we assume that the context has some

[7] An alternative, syntactically closer to the discrete case, relies on the Dirac δ distribution, whose value is zero everywhere except when its argument is zero, and whose total mass sums to one. Thus we recover a non-zero result after integration:

$$\mathrm{PDF}_p = \lambda x.p(\lambda y.\delta(x - y))$$

[8] More accurately, we would take U to be uniform over a finite set, S_U. Thus we would define it as $U = \lambda k.\Sigma_{u \in S_U} k(u)$.

type $\kappa = \alpha_1 \times ... \times \alpha_n$:

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

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

$$L_0 : u \to (\kappa \to r) \to r$$
$$L_0(u) = K \star \lambda\kappa.\mathit{observe}([\![u]\!]^\kappa) \star \lambda\diamond.\eta(\kappa)$$

To simplify the presentation, we have used the notation U^* to stand for a distribution over utterances which has already incorporated a notion of cost.[9] In our final formulation, both L_1 and L_0 have the same type. There is thus a more general notion of "listener", corresponding to a family of maps from utterances to distributions over contexts (or, equivalently, joint distributions over world states and linguistic parameters).[10] Such listeners differ only in how they update the prior — the literal listener uses a literal interpretation, while the pragmatic listener uses a pragmatic interpretation. Such pragmatic interpretations arise from the the speaker model, which chooses utterances which best fit the state of the world and linguistic parameters that it wishes to communicate.[11]

In summary, we have a realization of RSA that is highly compositional, in two senses. First, the models themselves are assembled compositionally in terms of probabilistic programs and monadic combinators. Second, utterances, represented by logical formulae, are interpreted compositionally, and such formulae may be obtained from natural language sentences using standard compositional techniques. At the same time, the mathematical vocabulary for describing RSA models is one and the same as that for describing linguistic meanings.

7 Conclusion

Our aim has been to lay a strong foundation for compositional probabilistic semantics. Many details have been left out, including about how one might represent prior knowledge, concretely. Many possibilities arise here. For instance, one may follow machine-learning methods and use vectors to represent individuals [3], while predicates are represented by hyperplanes in the relevant space [2]. An alternative would encode prior knowledge in terms of the same logic used to represent meanings, i.e., as sets of formulae. One may then constrain distributions over contexts in terms of such formulae [12]. Following this route, one may obtain a seamless integration of Bayesian and logical representations of knowledge.

---

9 To implement the definition of cost employed by RSA models, for example, U^* could be $U \star \lambda u.\mathit{factor}(e^{-\alpha * C(u)}) \star \lambda\diamond.\eta(u)$, given some uniform distribution U.

10 Emerson [5] advocates yet a third approach to RSA, in which linguistic parameters are marginalized out in the listener model altogether.

11 Systematically, if α tends to ∞; probabilistically, otherwise.

We should note that, while the logical fragments provided here are rudimentary, they are also merely expository: there is no deep reason that we did not provide a richer semantics for natural language expressions, e.g., incorporating dynamism (following a tradition of combining dynamic semantics with typed λ-calculus). Indeed, one could combine the framework we have illustrated with a logical semantics that *itself* uses continuations [1, 4, 16].

Finally, while our contribution is chiefly a theoretical one, the core aspects of the system described in this paper has been implemented using the Haskell programming language.[12] The mathematical vocabulary that we have employed here to assemble expressions of type r is closely mirrored by the implementation in terms of a domain-specific language for characterizing Markov chain Monte Carlo sampling procedures. Thus while many probabilistic programs cannot be evaluated to closed-form solutions, they may generally be finitely approximated, given sufficiently many samples. Most important, however, the modular representation of logical meaning and probabilistic side effects is straightforward to encode in Haskell, given the pure functional setting it provides.

We have shown how a probabilistic semantics of natural language is amenable to a fully formal treatment — one which remains squarely within the realm of pure typed λ-calculi. The key idea is to use an effect system to capture probabilistic operations (i.e., sampling and marginalization). Our approach fits the general framework of monadic semantics, and, as such, augments a literature that has grown in many exciting ways since the work of Shan [18].

[12] Available at https://github.com/juliangrove/grove-bernardy-lenls18.

Bibliography

[1] Barker, C., Shan, C.c.: Continuations and natural language, vol. 53. Oxford studies in theoretical linguistics (2014)

[2] Bernardy, J.P., Blanck, R., Chatzikyriakidis, S., Lappin, S., Maskharashvili, A.: Predicates as Boxes in Bayesian Semantics for Natural Language. In: Proceedings of the 22nd Nordic Conference on Computational Linguistics, pp. 333–337, Linköping University Electronic Press, Turku, Finland (Sep 2019), URL https://www.aclweb.org/anthology/W19-6137

[3] Bernardy, J.P., Blanck, R., Chatzikyriakidis, S., Maskharashvili, A.: Bayesian Natural Language Semantics and Pragmatics. In: Bernardy, J.P., Blanck, R., Chatzikyriakidis, S., Lappin, S., Maskharashvili, A. (eds.) Probabilistic Approaches to Linguistic Theory, CSLI Publications (2022)

[4] Charlow, S.: On the semantics of exceptional scope. PhD Thesis, NYU, New York (2014), URL https://semanticsarchive.net/Archive/2JmMWRjY

[5] Emerson, G.: Probabilistic Lexical Semantics: From Gaussian Embeddings to Bernoulli Fields. In: Bernardy, J.P., Blanck, R., Chatzikyriakidis, S., Lappin, S., Maskharashvili, A. (eds.) Probabilistic Approaches to Linguistic Theory, CSLI Publications (2022)

[6] Girard, J.Y.: Interprétation fonctionnelle et élimination des coupures de l'arithmétique d'ordre supérieur. Ph.D. thesis, Université Paris 7 (1972)

[7] Goodman, N.D., Frank, M.C.: Pragmatic Language Interpretation as Probabilistic Inference. Trends in Cognitive Sciences **20**(11), 818–829 (Nov 2016), ISSN 1364-6613, https://doi.org/10.1016/j.tics.2016.08.005, URL https://www.sciencedirect.com/science/article/pii/S136466131630122X

[8] Goodman, N.D., Lassiter, D.: Probabilistic Semantics and Pragmatics Uncertainty in Language and Thought. In: Lappin, S., Fox, C. (eds.) The Handbook of Contemporary Semantic Theory, pp. 655–686, John Wiley & Sons, Ltd (2015), ISBN 978-1-118-88213-9, https://doi.org/10.1002/9781118882139.ch21, URL http://onlinelibrary.wiley.com/doi/abs/10.1002/9781118882139.ch21, section: 21 _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781118882139.ch21

[9] Goodman, N.D., Mansinghka, V.K., Roy, D., Bonawitz, K., Tenenbaum, J.B.: Church: a language for generative models. In: Proceedings of the Twenty-Fourth Conference on Uncertainty in Artificial Intelligence, pp. 220–229, UAI'08, AUAI Press, Arlington, Virginia, USA (Jul 2008), ISBN 978-0-9749039-4-1

[10] Goodman, N.D., Stuhlmüller, A.: Knowledge and Implicature: Modeling Language Understanding as Social Cognition. Topics in Cognitive Science **5**(1), 173–184 (2013), ISSN 1756-8765, https://doi.org/https://doi.org/10.1111/tops.12007, URL https://onlinelibrary.wiley.com/doi/abs/10.1111/tops.12007

[11] Grice, H.P.: Logic and Conversation. In: Cole, P., Morgan, J.L. (eds.) Syntax and Semantics, vol. 3, Speech Acts, pp. 41–58, Academic Press, New York (1975)

[12] Grove, J., Bernardy, J.P., Chatzikyriakidis, S.: From compositional semantics to Bayesian pragmatics via logical inference. In: Proceedings of the 1st and 2nd Workshops on Natural Logic Meets Machine Learning (NALOMA), pp. 60–70, Association for Computational Linguistics, Groningen, the Netherlands (online) (Jun 2021), URL https://aclanthology.org/2021.naloma-1.8

[13] Jansson, P., Ionescu, C., Bernardy, J.P.: Probability Theory. In: Domain Specific Languages of Mathematics, pp. 223–246, no. 24 in Texts in computing (2022)

[14] Lassiter, D., Goodman, N.D.: Context, scale structure, and statistics in the interpretation of positive-form adjectives. Semantics and Linguistic Theory **23**(0), 587–610 (Aug 2013), ISSN 2163-5951, https://doi.org/10.3765/salt.v23i0.2658, URL https://journals.linguisticsociety.org/proceedings/index.php/SALT/article/view/2658, number: 0

[15] Lassiter, D., Goodman, N.D.: Adjectival vagueness in a Bayesian model of interpretation. Synthese **194**(10), 3801–3836 (Oct 2017), ISSN 1573-0964, https://doi.org/10.1007/s11229-015-0786-1, URL https://doi.org/10.1007/s11229-015-0786-1

[16] Lebedeva, E.: Expressing discourse dynamics through continuations. phdthesis, Université de Lorraine (Apr 2012), URL https://tel.archives-ouvertes.fr/tel-01749193

[17] Mohammed Ismail, W., Shan, C.c.: Deriving a probability density calculator (functional pearl). In: Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming, pp. 47–59, ICFP 2016, Association for Computing Machinery, New York, NY, USA (Sep 2016), ISBN 978-1-4503-4219-3, https://doi.org/10.1145/2951913.2951922, URL https://doi.org/10.1145/2951913.2951922

[18] Shan, C.c.: Monads for natural language semantics. arXiv:cs/0205026 (May 2002), URL http://arxiv.org/abs/cs/0205026, arXiv: cs/0205026