A LOGIC WITH MEASURABLE SPACES FOR NATURAL LANGUAGE SEMANTICS *

Jean-Philippe Bernardy   Rasmus Blanck
Aleksandre Maskharashvili

University of Gothenburg, Department of Philosophy, Linguistics and Theory of Science, Centre for Linguistics and Studies in Probability.
name.surname@gu.se

Abstract

We present a Logic with Measurable Spaces (LMS) and argue that it is suitable to represent the semantics of a number of natural language phenomena. LMS draws inspiration from several sources. It is decidable (like descriptive logics). It features Sigma spaces (like Martin-Löf type-theory). It internalises the notion of the cardinality (in fact, here, measures) of spaces (see [6]) and ratios thereof, allowing to capture the notion of event probability. In addition to being a powerful system, it is also concise and has a precise semantics in terms of integrals. Thanks to all these qualities, we hope that LMS can play a role in the foundations of natural language semantics.

1 Introduction

The ability of humans to reason under uncertainty has reflections within natural language where we find various lexico-syntactic constructions which allow us to express uncertain information. Moreover, we are able draw conclusions - make inferences under uncertainty. To give an adequate account to this crucial aspect of natural language, it has been long argued for employing probabilistic tools in defining semantics of natural language.

The question remains of which tool is best suited for the purpose. [7, 10] have proposed to use probabilistic programming languages.

In this paper, we propose to use instead a special-purpose language which aims at providing the most convenient interface for compositional semantics, while giving a fully precise semantics.

We call this intermediate language Logic with Measurable Spaces (LMS). We argue that LMS is suitable to represent the semantics of a number of natural language phenomena. LMS draws inspiration from several sources. It is decidable (like descriptive logics). It features Sigma spaces (like

*Supported by Swedish Research Council, Grant number 2014-39

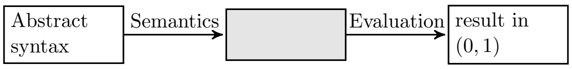

Figure 1: Overview of the parts of a complete probabilistic (bayesian) inference system

Martin-Löf type-theory). It internalises the notion of the cardinality (in fact, here, measures) of spaces (see [6]) and ratios thereof, allowing to capture the notion of event probability.

A fully-fledged probabilistic semantics will be comprised of several parts, shown in 1. In this paper, we focus on the interface between compositional semantics and evaluation *only*. Yet, to get a sense of how such a system is articulated, we present an example inference — remaining at a suitably abstract level:

$$\frac{\text{Most birds fly}}{\text{A few birds fly}}$$

We wish to compute the probability

$$P(\text{A few birds fly} \mid \text{Most birds fly})$$

and test if its value is closer to 1 than 0.

In order to do so, the natural sentences are first parsed, yielding abstract syntax trees. For example one can use the GF tool [11], but any tool which produces a syntax compatible with Montague-style categories would be suitable. The abstract syntax that we obtain for the premiss and the hypothesis could be written as follows.

$$P = many\ bird\ fly$$
$$H = aFew\ bird\ fly$$

The abstract syntax is then translated to LMS. This makes explicit their spaces, and the measures thereof. Using these features, the probabilities of all propositions of interest can be expressed precisely. To convert to this intermediate representation, we first must express our (lack of) prior knowledge about the common nouns, verbs, etc. present in the problem. To do so we gather the lexical items and introduce them as as random variables in the appropriate space. The premiss(es) are added as extra conditions, using a compositional semantics [2]. These conditions effectively update the distributions of the representations of lexical items, yielding a global space of situations Ω and assuming a proportion Θ_m corresponding to the meaning of "most".

$$\begin{aligned}\Omega = [&bird : Pred\\ &fly : Pred\\ &p : \mathsf{measure}([x : Ind; b : bird(x); f : fly(x)])\\ &\ > \Theta_m \mathsf{measure}([x : Ind; b : bird(x)])]\end{aligned}$$

Note that to make the language more concise, we unify the language of spaces and the language of propositions — effectively we sample p over the space of proofs of the propositions. We leave here the space of predicates $Pred$ abstract: possible choices are spelled out by [3] and [4].

The truth value of the conclusion is then expressed as a probability measure of a proposition over the whole space Ω that we just defined, with a suitable proportion Θ_f for "few".

$$X = P_{\omega:\Omega}([x : Ind; b : \omega.bird(x); f : \omega.fly(x)])$$

$$> \Theta_f \mathsf{measure}([x : Ind; b : \omega.bird(x)])$$

The convenient expression above can be turned into a mathematical expression using the semantics for spaces and probabilities (1). In our running example, the expression begins with integration over the spaces of predicates:

$$\sum_{bird:Pred} \sum_{fly:Pred} \frac{\mathbf{1}(P \wedge Q)}{\mathbf{1}(P)}$$

where the conditions P and Q are given by

$$\begin{aligned}P &= \mathsf{measure}([x : Ind; b : bird(x); f : fly(x)])\\ &\ > \Theta_m \cdot \mathsf{measure}([x : Ind; b : bird(x)])\\ Q &= \mathsf{measure}([x : Ind; b : bird(x); f : fly(x)])\\ &\ > \Theta_f \cdot \mathsf{measure}([x : Ind; b : bird(x)])\end{aligned}$$

The integrations and measures get further expanded if $Pred$ is made concrete. But, we can already see that $P \wedge Q = P$ if $\Theta_m > \Theta_f$, and in this simple case the integral therefore evaluates to 1, meaning that the inference is (stochastically) certain.

However, in the vast majority of cases, integrals are not computable symbolically. This would happen for example if we do not choose a fixed value Θ_m or Θ_f, but rather used random variables. In this kind of situation, one typically resorts to simulated sampling — using Monte Carlo methods (see 2.3).

$$
\begin{array}{rll}
A, B, \ldots ::= & \mathsf{IsTrue}(\phi) & \text{types of proofs} \\
 & |\Sigma(x : A)B & \text{sigma type, generalised pair} \\
 & |\mathsf{Distr}(d) & \text{Real-valued base distribution} \\
 & & \text{with finite support} \\
 & |\,\{e \,|\, x : A\} & \text{image of } A \text{ under } \lambda x.e \\
\phi, \psi, e ::= & x & \text{variable} \\
 & |\phi \wedge \psi & \\
 & |e_1 \leq e_2 & \\
 & |\pi_1(e)|\pi_2(e) & \text{projections} \\
 & |op(e_i) & \text{arithmetic operators} \\
 & |\Diamond & \text{uninformative object} \\
 & |\mathsf{measure}(A) & \text{internalisation of measure} \\
\tau, \sigma ::= & Unit|Bool|\mathbb{R}|\tau \to \sigma|\tau \times \sigma &
\end{array}
$$

Figure 2: Syntax of LMS

2 Logic with Measurable Spaces

In this section we describe a Logic with Measurable Spaces (LMS). LMS is the representation language connecting parse structures to mathematical expressions of probabilities. We use it to describe the meaning of inferences. As a first approximation, one can see LMS as a precise formalisation of informal notations used when manipulating logical expressions involving random variables. Readers familiar with these concepts can skip this section on first reading. But it will be helpful for understanding subsequent definitions.

The syntax of LMS is comprised of two categories: spaces (A, B, C, etc.), and expressions (e or ϕ, ψ for boolean-valued expressions.)

The main objects of interest are *spaces.* Every space has two aspects: an underlying *type* and a probability distribution over it. The types are formed by the unit type, booleans, reals, functions, and products.

In LMS, types are used as in a programming language, to verify that nonsensical expressions are disallowed. We do not follow the tradition of intuitionistic logic in that we ignore the inhabitants of types. Specifically, we do not consider types as propositions, via the Curry-Howard isomorphism. LMS does not include quantification over all types, nor over all spaces. Instead, the densities of spaces are their logical content. Before turning to

$$\frac{\Gamma \vdash \phi : Bool}{\Gamma \vdash \mathsf{IsTrue}(\phi) : \mathsf{Space}\, Unit} \qquad \frac{\Gamma \vdash A : \mathsf{Space}\,\tau \qquad \Gamma, x : \tau \vdash B : \mathsf{Space}\,\sigma}{\Gamma \vdash \Sigma(x : A)B : \mathsf{Space}\,(\Sigma(x : \tau)\sigma)}$$

$$\frac{\Gamma \vdash e_i : \mathbb{R}}{\Gamma \vdash \mathsf{Distr}(d_1[e_i]) : \mathsf{Space}\,\mathbb{R}} \qquad \frac{\Gamma, x : \tau \vdash e : \sigma \qquad \Gamma \vdash A : \mathsf{Space}\,\tau}{\Gamma \vdash \{e \,|\, x : A\} : \mathsf{Space}\,\sigma}$$

$$\frac{\Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \sigma[e_1/x]}{\Gamma \vdash (e_1, e_2) : \tau \times \sigma} \qquad \frac{\Gamma \vdash e : \tau \times \sigma}{\Gamma \vdash \pi_1(e) : \tau} \qquad \frac{\Gamma \vdash e : \tau \times \sigma}{\Gamma \vdash \pi_2(e) : \sigma}$$

$$\Gamma \vdash \Diamond : Unit \qquad \frac{\Gamma \vdash \phi : Bool \qquad \Gamma \vdash \psi : Bool}{\Gamma \vdash \phi \wedge \psi : Bool} \qquad \frac{\Gamma, x : \tau \vdash e : \sigma}{\Gamma \vdash \lambda x.e : \tau \to \sigma}$$

$$\frac{\Gamma \vdash e_0 : \tau \to \sigma \qquad \Gamma \vdash e_1 : \tau}{\Gamma \vdash e_0(e_1) : \sigma} \qquad \Gamma \vdash true : Bool \qquad \Gamma \vdash false : Bool$$

$$\frac{\Gamma \vdash e : Bool}{\Gamma \vdash \mathbf{1}(e) : \mathbb{R}} \qquad \Gamma \vdash k : \mathbb{R} \qquad \frac{\Gamma \vdash e_i : \mathbb{R}}{\Gamma \vdash op(e_i) : \mathbb{R}}$$

Figure 3: Typing rules for LMS. In the above *op* stands for an arbitrary arithmetic operator or arbitrary arity, with e_i being its operands. Similarly, we list only only one logical connective ($\wedge$); others follow the same pattern.

density we give a brief overview of LMS typing and its consequences. We use two judgments. First, the judgment $\Gamma \vdash e : \tau$, which is the standard typing judgment for terms in the simply typed lambda calculus. We call Boolean-valued expressions *propositions*, so if $\Gamma \vdash \phi : Bool$ holds, it means that ϕ is a proposition in context Γ. Second, the judgment $\Gamma \vdash A : \mathsf{Space}\,\sigma$ expresses that A is a space over the ground type σ, in a context Γ.

Because expressions are simply typed, they inherit the usual normalisation properties of typed lamdba terms [1]. Any closed term of type $\mathbb{R}$ is a real number.

We now focus on spaces and distributions over them. We have four basic space constructions:

1. Given a distribution with n parameters $d(x_1, \ldots, x_n)$, we have the space $\mathsf{Distr}(d_1(e_1, \ldots, e_n))$ (each of the parameters can be assigned any real-valued expression).

2. We can construct a space whose density is 1 when a proposition ϕ is

true and 0 otherwise. It is written $\mathsf{IsTrue}(\phi)$.

3. We can construct sigma spaces. Given a space A and a space $B[x]$, we can write $\Sigma(x : A)B[x]$ for the the sigma space.

4. We can take the image of a space A under a function f. This space is written $\{f(x) \mid x : A\}$. (In fact we generalise to and allow any expression dependent on x instead of just $f(x)$.)

These constructions are listed in 3.

Formally, we do not manipulate densities directly, thus avoiding theoretical difficulties, in particular for $\{f(x) \mid x : A\}$. Instead, we generalise the notion of integration so that it does not just apply to distributions, but to arbitrary spaces. For this purpose we use the symbol $\sum$, as it is a natural extension of the summation operator.

Definition 1. If $\Gamma \vdash A : \mathsf{Space}\,\alpha$ and $\Gamma, x : \alpha \vdash e : \mathbb{R}$, we define $\sum_{x:A} e$ (which can be read as the integral of e for x ranging over A), by induction on A:

$$\begin{aligned}
\sum_{x:\mathsf{Distr}(d)} e &= \int_{\mathbb{R}} \mathrm{PDF}(d, x) \cdot [\![e]\!] dx \\
\sum_{x:\mathsf{IsTrue}(\phi)} e &= \mathbf{1}([\![\phi]\!]) \cdot [\![e[\Diamond/x]]\!] \\
\sum_{z:\Sigma(x:A)B} e &= \sum_{x:A} \sum_{y:B} e[(x, y)/z] \\
\sum_{y:\{e \mid x:A\}} e_2 &= \sum_{x:A} e_2[e/y]
\end{aligned}$$

Definition 2. (Evaluation of expressions) The value of an expression e is written $[\![e]\!]$ and defined by induction on the structure of expressions, as is standard in the lambda calculus. We know that evaluation terminates because of our type-system. The only case that merits attention is the evaluation of $\mathsf{measure}(A)$, which is specified by the following equation:

$$[\![\mathsf{measure}(A)]\!] = \sum_{x:A} 1$$

The expression $\sum_{x:A} 1$ integrates over the whole space of the constant value 1, thus "counting" the elements of that space. Therefore it is the *measure* of the space A. Overloading the notation, we also write $\mathsf{measure}(A)$ for the measure of the space A as a meta-theoretical expression (not an LMS expression), with the same definition.

Definition 3. (Expected value) We define the *expected value* of e for a random variable x distributed in A as follows:

$$E_{x:A}(e) = \frac{\sum_{x:A} e}{\mathsf{measure}(A)}$$

Remark:

$$E_{z:(\Sigma(x:A)B)}(e) = E_{x:A}(E_{y:B}(e[(x, y)/z]))$$

Notation:

$$E_{x_0:A_0,\ldots,x_n:A_n}(e) = E_{x_0:A_0}(\ldots E_{x_n:A_n}(e))$$

Finally, we can define the *probability* of a proposition ϕ over a random variable x ranging in A as the proportion of (the measure of) the space A where ϕ holds.

Definition 4.

$$P_{x:A}(\phi) = E_{x:A}(\mathbf{1}(\phi))$$

An equivalent definition is the following:

$$P_{x:A}(\phi) = \frac{\mathsf{measure}(\Sigma(x:A)\mathsf{IsTrue}(\phi))}{\mathsf{measure}(A)}$$

In general, for probabilistic inference, we define a space of possible situations Ω, and evaluate the expected truth value of some proposition ϕ over this space. The space Ω typically has a complex structure.

We now verify that $P_{x:A}(\phi)$ satisfies the expected properties of probabilities, starting with the following lemma:

Lemma 1. $\sum_{x:A}$ *is a linear operator*

(i) $\sum_{x:A}(k \cdot t) = k \cdot \sum_{x:A} t$ *if k does not depend on x*

(ii) $\sum_{x:A}(t + u) = \sum_{x:A} t + \sum_{x:A} u$

Proof. By induction on the structure of A. □

When a space A has zero measure, the probabilities over it are undefined. Otherwise, the Kolmogorov laws of probability are respected. It is easy to verify that any probability is positive, and that the probability of *true* is 1. The last law (in its finite variant) needs a bit more work, and its proof follows.

Theorem 1. *If $\phi \wedge \psi = false$, then*

$$P_{x:A}(\phi \vee \psi) = P_{x:A}(\phi) + P_{x:A}(\psi)$$

Proof.

$$\begin{aligned} E_{x:A}(\phi \vee \psi) &= \sum_{x:A} \mathbf{1}(\phi \vee \psi) && \text{by def.} \\ &= \sum_{x:A} (\mathbf{1}(\phi) + \mathbf{1}(\psi)) && \text{because } \phi \wedge \psi = false \\ &= \sum_{x:A} \mathbf{1}(\phi) + \sum_{x:A} \mathbf{1}(\psi) && \text{by linearity of } \sum_{x:A} \\ &= E_{x:A}(\phi) + E_{x:A}(\psi) && \text{by def.} \end{aligned}$$

The result is obtained by dividing by $\mathsf{measure}(A)$. □

The property that probabilities are positive can be checked in a similar way. The assumption of unit measure ($P_{x:A}(true) = 1$) is a simple consequence of the definition.

2.1 Dealing with equality

In some situations it is useful to use equality of real-valued expressions (for example “john is as tall as mary”). Perhaps the most obvious way to encode equality between x and y is by using $\mathsf{IsTrue}(x = y)$. Assume that x and y are both taken in a space A of strictly positive measure, we can naively write the space B of equal x and y as follows.

$$B = \Sigma(x : A)\Sigma(y : A)\mathsf{IsTrue}(x = y)$$

Unfortunately, the above definition is problematic, because $x = y$ is stochastically impossible for real-valued x and y. [1] Consequently $\mathsf{measure}(B) = 0$. In turn, when evaluating probabilities involving B, one gets division by zero and the probabilities are undefined using the definitions given above.

2.1.1 A theoretical approach

What we would like is to replace $\mathsf{IsTrue}(x = y)$ by another space $x \equiv y$, such that the density of $x \equiv y$ is zero when $x \neq y$, but whose total measure is 1 (instead of 0). This can be done conceptually by increasing the density at the points where $x = y$. To do this, we must first introduce the $\mathsf{Factor}(e)$ space, which acts like $\mathsf{IsTrue}(\phi)$, but e gives directly the factor to be used

[1] Readers who are not familiar with this property can convince themselves informally by seeing that getting x and y to be equal requires an impossible alignment of infinite precision. Formally this can be seen by carrying out the computation of integrals as defined above.

in the integration (which can thus be greater than 1). Hence, its integrator is as follows:

$$\sum_{x:\mathsf{Factor}(e_1)} e_2 = [\![e_1]\!] \cdot [\![e_2[\Diamond/x]]\!].$$

Second, we need to pick a sufficiently great factor, so when integrating it over a 0-measure area, the result end up being 1. This can only be done with an infinitely large factor.

One may believe that no such space exists, but, fortunately, such a space has already been extensively studied, and it is known as the *Dirac δ function*. Classically, δ has a single parameter, and its density is 0 when this parameter is nonzero, and its defining property is:

$$\int_{-\infty}^{\infty} f(x)\delta(x)\,dx = f(0)$$

In terms of spaces, the same property becomes:

$$\sum_{x:\mathsf{Distr}(\delta)} t = t[0/x]$$

Hence we can define $x \equiv y$ to be for $\mathsf{Factor}(\delta(x-y))$.

We can now compute the measure of our motivating example B:

$$\begin{aligned}
\mathsf{measure}(\Sigma(x:A)\Sigma(y:A)x \equiv y) &= \sum_{x:A}\sum_{y:A}\sum_{p:\mathsf{Factor}(\delta(y-x))} 1 \\
&= \sum_{x:A}\sum_{z:\{y-x\,|\,x:A\}}\sum_{p:\mathsf{Factor}(\delta(z))} 1 \quad \text{by substitution} \\
&= \sum_{x:A}\sum_{z:\{y-x\,|\,x:A\}}\sum_{z:\mathsf{Distr}(\delta)} 1 \\
&= \sum_{x:A}\sum_{z:\{y-x\,|\,x:A\}} 1 \quad \text{by } \delta \text{ property} \\
&= \sum_{x:A}\sum_{y:A} 1 \\
&= \mathsf{measure}(A)^2
\end{aligned}$$

We see that involving $x \equiv y$ does not make the measure of spaces 0, and hence probabilites remain well-defined. Computing symbolic integration involving δ is not possible in every case, but we refer the reader to [12] for a generic method.

2.1.2 A numerical approach

Perhaps more disturbing that δ not being always computable, it does not lend itself well to Monte Carlo methods, which we describe in 2.3. We essentially are faced with the same problem as originially. If we sample random any x and y, and their numerical representations have a high resolution then, it will be extremely rare that $x = y$, and the Monte-Carlo approximation will not converge.

A possible solution is to increase the density in a non-zero region around the points such that $x = y$, in a smooth fashion. One way to do that is to take the density of the space $x \equiv y$ to be a Gaussian curve of a suitably small standard deviation σ[2] and which has its maximum at $x = y$:

$$\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{(x-y)}{\sigma})^2}$$

Like all probability density functions, the gaussian has density 1, and we thereby avoid spaces of zero measure.

While this approach is pleasing, choosing a suitable value for σ is not necessarily obvious. If it is too small, then we fall into the original pitfall: most of the time the density will be too small to contribute significantly to the integral. Conversely, if σ is too large, then one gets an excessively imprecise result. Unless otherwise stated, we have run our models with $\sigma = 1$.

2.2 Record notation

When dealing with complex structures involving nested Σ spaces, the expressions for projections become quickly inscrutable. For this reason we use the record notation for such spaces and the corresponding projections.

Definition 5. (Record spaces and projections) Formally, record spaces are defined by translation to Σ spaces, as follows:

$$[x_1 : A_1; \ldots ; x_n : A_n] = \Sigma(x_1 : A_1)\Sigma(x_2 : A_2) \ldots A_n$$

Additionally if $e : [x_1 : A_1; \ldots ; x_n : A_n]$, then $e.x_i$ is a shorthand for $\pi_1(\pi_2(\pi_2(\ldots e)))$ (the number of repetitions of π_2 is the index of the field in the record).

For similar reasons, we use a shorthand notation for the expected value over several variables, defined as follows:

$$E_{x_0:A_0,\ldots,x_n:A_n}(e) = E_{x_0:A_0}(\ldots(E_{x_n:A_n}(e)))$$

[2] In fact, if f_σ is a gaussian function with mean 0 and standard deviation σ, then $\delta(x) = \lim_{\sigma\to 0} f_\sigma(x)$

2.3 Approximation via sampling

Unfortunately, in the majority of cases the mathematical expressions produced by the semantics given above contain integrals which cannot be evaluated symbolically.

Hence, we are forced to resort to a numerical approximation algorithm to evaluate them. We use a variant of Gibbs sampling, which is itself an instance of a Markov Chain Monte Carlo (MCMC) method. The algorithm that we use closely follows the one described in [9].

All Monte Carlo methods are based on the same principle, which can be outlined as follows.To evaluate $P_{x:A}(\phi[x])$: 1. Sample a random x in A; 2. Check if $\phi[x]$ holds for a chosen value of x; 3. Repeat this process a large number of times.The ratio of the number successes to the number of tries converges to $P_{x:A}(\phi[x])$ as the number of tries tends to infinity.

In certain cases it is very hard to find any sample $x : A$. If (say) A contains an $\mathsf{IsTrue}(\psi)$ space where ψ is satisfied one time in a million, then it will be necessary to try a million samples until one try can be counted. In our application, these kind of situations will happen whenever 1. sets with many hypotheses are considered, 2. very strong hypotheses are tested. For example, “99.9 percent of men walk” requires such a precise arrangement of parameters that most samples will end up being discarded when this condition is checked.

To mitigate this problem, MCMC methods do not sample elements independently. Rather, each new sample x is based on a previous sample. Typically, only a single parameter is changed at every step. On average, the next sample is chosen to be as probable as the previous one, or more so. This way, the system is able to find many (probable) samples.

But samples can form (probably) disconnected regions in the chain space, and thus certain configurations may end up being explored more thoroughly than other, equally (or more) probable ones.

Ultimately, it is up to the designer of the underlying problem to avoid the pitfalls of the approximation methods. Because the phrasing of the hypotheses are infinitely variable for any natural language, we cannot avoid these pitfalls entirely. However, certain semantic designs will be more prone to problems than others.

3 Quantifiers

Even though it has very few constructions, LMS is sufficently expressive to encode the usual logical quantifiers: every x in A satisfies ϕ iff the subspace of A where ϕ holds is (at least) as big as A itself.[3] The definition of the existential quantifier follows a similar pattern.

$$\forall x : A.\phi \overset{def}{=} \mathsf{measure}(A) \leq \mathsf{measure}(\Sigma(x : A).\mathsf{IsTrue}(\phi))$$
$$\exists x : A.\phi \overset{def}{=} 0 \leq \mathsf{measure}(\Sigma(x : A).\mathsf{IsTrue}(\phi))$$

4 Comparison with probabilistic programming languages

With LMS, we propose a way to describe types and an associated density (spaces). The tradition in the linguistic community is to use instead probabilistic programming languages [8, 5, 9]. Simply put, probabilistic programming languages do not describe spaces as such, but instead functions which generate elements of a certain type. Using LMS presents advantages. First, probabilistic programming languages typically do not natively offer the option to run an inference *within* another inference. In contrast this is done straightforwardly in LMS using the $\mathsf{measure}(e)$ expression. Second, the semantics of LMS is more straightforward than that of a formal probabilistic programming language: this is because LMS does not allow sampling within expressions. (Only spaces can refer to other spaces). We refer the reader to the work of [5] for an example of a probabilistic programming language equipped with formal semantics. Third, constructing spaces is very similar to constructing types and logical formulas. Thus we hope that LMS can readily be used by linguists who are used to interpret natural language into type theories (or similar logical systems).

5 Conclusion

In sum, LMS aims to solve a language design problem. It aims to bridge a semantic gap between abstract syntax for natural languages and the evaluation of probabilistic truth values.

On the one hand, this language is sufficiently powerful to express probabilistic problems, is convenient enough to support probabilistic syllogisms. On the other hand, it has a simple model in terms of integrators, and, for linguistic purposes it compares favourably with usual probabilistic programming languages.

[3] This definition is problematic when ϕ logically false for some x, but still stochastically true. To deal with such cases, one must then use disintegrators, as explained by [12].

References

1. Barendregt, H. P. Lambda calculi with types. *Handbook of logic in computer science 2* (1992), 117–309.

2. Bernardy, J.-P., Blanck, R., Chatzikyriakidis, S., and Lappin, S. A compositional Bayesian semantics for natural language. In *Proceedings of the International Workshop on Language, Cognition and Computational Models, COLING 2018, Santa Fe, New Mexico* (2018), pp. 1–11.

3. Bernardy, J.-P., Blanck, R., Chatzikyriakidis, S., Lappin, S., and Maskharashvili, A. Bayesian inference semantics: A modelling system and a test suite. In *Proceedings of the Eighth Joint Conference on Lexical and Computational Semantics (*SEM), Minneapolis* (2019), Association for Computational Linguistics, pp. 263–272.

4. Bernardy, J.-P., Blanck, R., Chatzikyriakidis, S., Lappin, S., and Maskharashvili, A. Predicates as boxes in bayesian semantics for natural language. In *Proceedings of the 22nd Nordic Conference on Computational Linguistics* (2019), ACL.

5. Borgström, J., Gordon, A. D., Greenberg, M., Margetson, J., and Van Gael, J. Measure transformer semantics for Bayesian machine learning. *Logical Methods in Computer Science 9* (2013), 1–39.

6. Fox, C., and Lappin, S. *Foundations of Intensional Semantics.* Blackwell, 2005.

7. Goodman, N., and Lassiter, D. Probabilistic semantics and pragmatics: Uncertainty in language and thought. In *The Handbook of Contemporary Semantic Theory, Second Edition*, S. Lappin and C. Fox, Eds. Wiley-Blackwell, Malden, Oxford, 2015, pp. 655–686.

8. Goodman, N., Mansinghka, V. K., Roy, D., Bonawitz, K., and Tenenbaum, J. Church: a language for generative models. In *Proceedings of the 24th Conference Uncertainty in Artificial Intelligence (UAI).* 2008, pp. 220–229.

9. Goodman, N., and Stuhlmüller, A. The Design and Implementation of Probabilistic Programming Languages. `http://dippl.org`, 2014. Accessed: 2018-4-17.

10. Lassiter, D., and Goodman, N. Adjectival vagueness in a Bayesian model of interpretation. *Synthese 194* (2017), 3801–3836.

11. Ranta, A. Grammatical framework. *Journal of Functional Programming 14*, 2 (2004), 145–189.

12. Shan, C.-c., and Ramsey, N. Exact bayesian inference by symbolic disintegration. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages* (2017), POPL, pp. 130–144.