# Dependent Type Networks: A Probabilistic Logic via the Curry-Howard Correspondence in a System of Probabilistic Dependent Types

**Jonathan Warrell**[1,2,*] **Mark Gerstein**[1,2,3]

[1]*Program in Computational Biology and Bioinformatics, Yale University, New Haven, CT 06520, USA*
[2]*Department of Molecular Biophysics and Biochemistry, Yale University, New Haven, CT 06520, USA*
[3]*Department of Computer Science, Yale University, New Haven, CT 06520, USA*
*jonathan.warrell{@yale.edu, @gmail.com}

## Abstract

We first introduce a *Probabilistic Dependent Type System* (PDTS) via a functional language based on a subsystem of intuitionistic type theory including dependent sums and products, which is expanded to include stochastic functions. We provide a sampling-based semantics for the language based on non-deterministic beta reduction. Further, we formulate a probabilistic logic by defining a *Dependent Type Network* (DTN) within the PDTS, where the DTN is derived as a direct result of the Curry-Howard correspondence. We show that the DTN framework is equivalent to Markov Logic Networks, and hence is a universal representation for discrete distributions.

## 1 Introduction

Stochastic functional programming languages have used non-deterministic beta reduction to provide a probabilistic semantics for languages based on the untyped lambda calculus [1] and probabilistic type systems based on the simply-typed lambda calculus [2]. Here, we show that a similar approach can be used to provide semantics for a probabilistic type system based on intuitionistic type theory [3], hence including also dependent types. We call these *Probabilistic Dependent Type Systems* (PDTS).

We demonstrate that a probabilistic logic can be naturally formulated in our PDTS framework, which we term a *Dependent Type Network* (DTN), using the Curry-Howard correspondence. The logic we derive is distinct from previous formulations of probabilistic logic: Previous approaches have involved associating weights with sentences which induce a distribution over interpretations (possible worlds) [4]; formulating conditions for consistently assigning probabilities to sentences or interpretations in a simply-typed higher-order language [5, 6]; or assigning terms and deterministic functions probabilistically to types via probabilistic type judgments [7]. In contrast, we formulate an intuitionistic type system of non-deterministic functions, and derive a probabilistic logic as a subsystem. We show that DTNs are equivalent to Markov Logic Networks (MLNs) [4], and hence are a universal representation for discrete distributions.

Sec. 2 describes our ambient Probabilistic Dependent Type System model, while Sec. 3 describes the embedded Dependent Type Network formulation of probabilistic logic. Sec. 4 summarizes the connection between DTNs and Markov Logic Networks, and Sec. 5 concludes with a discussion, including relationships to deep-learning and probabilistic programming methods. A more extended exposition is given in Supplementary Materials, and we refer to relevant sections throughout.

## 2 A System of Probabilistic Dependent Types

We begin by defining a probabilistic dependent type system $\lambda P\rho^{\Sigma\,\mathrm{bool}}$. Since this system includes stochastic functions and dependent types, we naturally derive expressions which represent distributions over terms of multiple types, distributions over types, as well as functions whose return type is probabilistically determined. We first define an abstract syntax for the language:

$$\begin{array}{lcl}
\mathcal{C} & ::= & 1 \mid \mathrm{true} \mid \mathrm{false} \mid \mathrm{unit} \mid \mathrm{bool} \\
\mathcal{S} & ::= & * \mid \Box \\
\mathcal{V} & ::= & \{a, b, c, ...A, B, C...a_1, b_1, c_1...a_2...\} \\
\mathcal{U} & ::= & \mathcal{C} \mid \mathcal{V} \mid (\mathcal{U}, \mathcal{U})_{\mathcal{T}} \\
\mathcal{Z} & ::= & \{\} \mid \{(\mathcal{T}, \mathcal{T}),\ \mathcal{Z}\} \\
\mathcal{T} & ::= & \mathcal{U} \mid (\mathcal{T}\mathcal{T}) \mid \lambda\mathcal{V} : \mathcal{T}.\mathcal{T} \mid \mathrm{random}_\rho(\mathcal{T}) \mid \\
 & & \mathrm{if}\,\mathcal{V}\,\mathrm{then}\,\mathcal{T}\,\mathrm{else}\,\mathcal{T} \mid \pi_1\mathcal{T} \mid \pi_2\mathcal{T} \mid \\
 & & \sum\mathcal{V} : \mathcal{T}.\mathcal{T} \mid \prod\mathcal{V} : \mathcal{T}.\mathcal{T} \mid (\lambda\mathcal{V}.\mathcal{Z})\mathcal{T}, \quad (1)
\end{array}$$

where $\mathcal{C}$ are type/value constants, $\mathcal{S}$ are sorts, $\mathcal{V}$ are variables, $\mathcal{U}$ are values, $\mathcal{T}$ are pseudo-expressions (a subset of which will be legal expressions), $\rho$ is any real number in the interval $[0\ 1]$, and the syntactic construction $\mathcal{Z}$ allows us to form lists of paired terms. We will refer to the deterministic subsystem built out of expressions not involving $\mathrm{random}$ and $(\lambda\mathcal{V}.\mathcal{Z})$ constructions as $\lambda P^{\Sigma\,\mathrm{bool}}$, and write $\mathcal{T}_{\lambda P^{\Sigma\,\mathrm{bool}}}$ for the set of common pseudo-expressions. Next, we define inductively a weighted $\beta$-reduction relationship ($\beta\rho$-reduction) on pseudo-expressions, $\tau_1 \rightarrow^{\rho}_{\beta} \tau_2$ via:

$$\begin{array}{rcl}
\pi_1(u_1, u_2)\mathcal{T} & \rightarrow^1_\beta & u_1 \\
\pi_2(u_1, u_2)\mathcal{T} & \rightarrow^1_\beta & u_2 \\
\text{if true then } \tau_1 \text{ else } \tau_2 & \rightarrow^1_\beta & \tau_1 \\
\text{if false then } \tau_1 \text{ else } \tau_2 & \rightarrow^1_\beta & \tau_2 \\
\mathrm{random}_\rho(\tau) & \rightarrow^\rho_\beta & (\tau\,\mathrm{true}) \\
\mathrm{random}_\rho(\tau) & \rightarrow^{(1-\rho)}_\beta & (\tau\,\mathrm{false}) \\
(\lambda x : \tau_1.\tau_2)\tau_3 & \rightarrow^1_\beta & \tau_2[x := \tau_3] \\
 & & \text{if } \tau_3 \in \mathcal{T}_{\lambda P^{\Sigma\,\mathrm{bool}}}, \\
 & & \text{and no FV captured,} \\
(\lambda x.Z)\tau_3 & \rightarrow^1_\beta & (\lambda x : \tau_1.\tau_2)\tau_3 \\
 & & \text{if } \tau_3 \in \mathcal{T}_{\lambda P^{\Sigma\,\mathrm{bool}}}, \\
 & & \tau_3 : \tau_1,\, (\tau_1, \tau_2) \text{ in } Z, \\
 & & \text{and no conflicting} \\
 & & \text{reducts of same type.}
\end{array} \tag{2}$$

where, $u_1, u_2 \in \mathcal{U}$, $\tau_1, \tau_2, \tau_3, \tau \in \mathcal{T}$, $b_1, b_2 \in \{\mathrm{true}, \mathrm{false}\}$, $Z \in \mathcal{Z}$, FV abbreviates free variables, and we write 'in' for the list membership relation (see Supp. Eqs. 3 and 6 for full description of FV and conflicting reducts conditions). The reduction relation $\beta\rho$ is extended by allowing reduction of sub-pseudo-expressions while enforcing a 'stochastic memoization' property as defined in [1], and we write $\tau_1 \twoheadrightarrow^{\rho_0}_\beta \tau_2$ when the sum of the products of the $\rho$ values across all reduction chains between $\tau_1$ and $\tau_2$ is $\rho_0$ (see Supp Eq. 6 and following, and Supp. Prop. 5).

To apply the reduction rules in Eq. 2, it is necessary to define the relation (or *judgement*) $a : A$, which can be read as '$a$ is of type $A$'. This can be done by specifying the type inference rules for the system. For the deterministic component, $\mathcal{T}_{\lambda P^{\Sigma\,\mathrm{bool}}}$, we use standard typing rules for dependent type systems (see [9] and Supp. Eq. 4). Note particularly, that an expression of the form $\prod a : A.B$ represents the dependent type of functions whose inputs are of type $A$, and whose outputs may have a type which depends on their input $B(a)$. Similarly, an expression of the form $\sum a : A.B$ represents the type of pairs whose first element is of type $A$, and whose second element has the dependent type $B(a)$. Also, note that all typing judgments are relative to a (possibly empty) *context*, $\Gamma$, which assigns types to any FVs. To identify legal expressions in $\lambda P\rho^{\Sigma\,\mathrm{bool}}$ (i.e. including non-deterministic expressions), we define two operators $\mathrm{Types}$ and $\mathrm{Reductions}$ on pseudo-expressions, where $\mathrm{Reductions}(\tau)$ returns the set of normal forms to which a given pseudo-expression reduces with non-zero probability, and $\mathrm{Types}(\tau)$ returns the set of types corresponding to those normal forms (we will ensure all reduction sequences of legal expressions in $\lambda P\rho^{\Sigma\,\mathrm{bool}}$ end in $\lambda P^{\Sigma\,\mathrm{bool}}$ normal forms). We define these operators inductively (see Supp. Eq. 7), and define the legal expressions in $\lambda P\rho^{\Sigma\,\mathrm{bool}}$ as all expressions $\tau$ with non-empty $\mathrm{Types}(\tau)$ and $\mathrm{Reductions}(\tau)$ sets.

We can demonstrate that $\lambda P\rho^{\Sigma\,\mathrm{bool}}$ has a number of desirable properties. First, we have that $\tau_1 \rightarrow^\rho_\beta \tau_2$, implies $\mathrm{Types}(\tau_1) \supseteq \mathrm{Types}(\tau_2)$, and $\mathrm{Reductions}(\tau_1) \supseteq \mathrm{Reductions}(\tau_2)$ (Supp. Prop. 3), hence $\beta\rho$-reduction weakly preserves types (*weak preservation*). Second, we can show that a *progress* property holds, in that for any legal expression the $\rho$ values associated with the available reductions sum to 1, implying that we can make a probabilistic reduction step hence giving sampling semantics to the relation $\beta\rho$ (Supp. Prop. 4a), and that all valid reduction sequences of expressions end in an expression of $\lambda P^{\Sigma\,\mathrm{bool}}$ in normal form (Supp. Prop. 4b). Further, we can show that $\mathrm{Reductions}(\tau)$ and $\mathrm{Types}(\tau)$ contain exactly the normal forms and associated types to which $\tau$ reduces with non-zero probability (Supp. Prop. 6). Together, these properties imply that *probabilistic type judgement* is possible by sampling in $\lambda P\rho^{\Sigma\,\mathrm{bool}}$. We summarize this in the following (writing NF for normal form):

**Theorem 1.** *(Probabilistic Type Judgment): Letting* $\iota_{\tau_1}(\tau_2) = \sum_{\{\tau_3 \in \mathrm{NF}_{\lambda P^{\Sigma\,\mathrm{bool}}} | \tau_3 : \tau_2\}} \rho_{\tau_3}$*, where* $\tau_1 \twoheadrightarrow^{\rho_{\tau_3}}_\beta \tau_3$*, we have (a)* $\iota_{\tau_1}(\tau_2) > 0 \Rightarrow \tau_2 \in \mathrm{Types}(\tau_1)$ *and* $\iota_{\tau_1}(\tau_2) = 0 \Rightarrow \tau_2 \notin \mathrm{Types}(\tau_1)$*; and (b)* $\iota_{\tau_1}(\tau_2)$ *can be estimated by the frequency with which* $\tau_1$ *reduces to* $\tau_3 \in \mathrm{NF}_{\lambda P^{\Sigma\,\mathrm{bool}}}$ *such that* $\tau_3 : \tau_2$.

A proof is provided in the supplement (Supp. Prop. 7).

## 3 Dependent Type Networks

We now formulate a *Dependent Type Network* (DTN), which will be a weighted network of formulae in first-order logic embedded in a PDTS as above. We require a base context $\Gamma_0$ which fixes symbols for the constants, predicates and functions of the model, for instance containing symbols (/types-constructors) $B^1_{n=1 \ldots N_{B^1}}$, and $B^2_{n=1 \ldots N_{B^2}}$ for unary and binary predicates respectively;

$b^1_{n=1...N_{B^1}}$, and $b^2_{n=1...N_{B^2}}$ for terms (/proofs) associated with these predicates; and $c_{1...N_c}$ for fixed constants (see Supp. Eq. 11 and following for a full definition of $\Gamma_0$). A DTN is fixed by augmenting this base context with a further set of 'formulas' in the language, specified by $\Gamma_1 = \Gamma_0, < f_1 : F_1, ..., f_{N_F} : F_{N_F} >$. Here, a given formula such as $B^1(c_1) \wedge B^1(c_2)$ is represented by the type corresponding to the formula by the Curry-Howard correspondence, $B^1(c_1) \times B^1(c_2)$. In general, $X \wedge Y$ corresponds to $X \times Y$ ($X$ and $Y$ being ground predicates), $X \vee Y$ to the *disjoint product* $\sum x : \text{bool}\,.(\text{if}\, x\, \text{then}\, X\, \text{else}\, Y)$, $X \rightarrow Y$ to $X \rightarrow Y$, $\neg X$ to $X \rightarrow \perp$, $\forall x (X(x))$ to $\prod x : A.X(x)$, and $\exists x(X(x))$ to $\sum x : A.X(x)$. An expression which can be typed to a formula type corresponds to a proof of that formula, and thus to asserting the formula itself (a formula is 'true' if it is inhabited). We will further equip a DTN with a set of probabilistic weights, $P = \{p_1, p_2, ..., p_{N_F}\}$, where $p_i$ will represent the probability that $F_i$ is true, taking values in $(0 \;\; 1)$ (see Supp. Def. 1 for a full definition of DTN).

Given a DTN as defined, we will represent a query formula, for instance $Q = B^1_1(c_3)$, as an expression (/program) $q_Q$ where $\text{Types}(q_Q) = \{B^1_1(c_3), B^1_1(c_3) \rightarrow \perp, \perp\}$. That is, the expression reduces to either a proof of $Q$, a proof of $\neg Q$, or a proof of a contradiction (the last possibility corresponding to an inconsistent world). We will canonically define $q_Q$ for a particular $Q$ (where $Q$ is a type) as follows:

$$\begin{aligned} q_Q \;=\;\; & \text{let}\, x_1 = \text{random}_{p_1}(\lambda x : \text{bool}\,.\,\text{if}\, x\, \text{then}\, f_1 \\ & \quad \text{else}\, 1)\,\text{in} \\ & ... \\ & \text{let}\, x_{N_F} = \text{random}_{p_{N_F}}(\lambda x : \text{bool}\,.\,\text{if}\, x\, \text{then}\, f_{N_F} \\ & \quad \text{else}\, 1)\,\text{in} \\ & \text{let}\, x_{N_F+1} = D\,\text{in} \\ & (\lambda x.Z_Q)(x_1, x_2, ..., x_{N_F}, x_{N_F+1}). \end{aligned} \tag{3}$$

In Eq. 3, $D$ is defined as follows:

$$\begin{aligned} D \;=\;\; & \text{let}\, x_{1,1} = \text{random}_{0.5}(\lambda x : \text{bool}\,.\,\text{if}\, x\, \text{then}\, b_{1,1,1} \\ & \quad \text{else}\, b_{1,1,0})\,\text{in} \\ & ... \\ & \text{let}\, x_{N_{B^1},N_C} = \text{random}_{0.5}(\lambda x : \text{bool}\,.\,\text{if}\, x\, \text{then} \\ & \quad b_{N_{B^1},N_C,1}\, \text{else}\, b_{N_{B^1},N_C,0})\,\text{in} \\ & \text{let}\, x_{1,1,1} = \text{random}_{0.5}(\lambda x : \text{bool}\,.\,\text{if}\, x\, \text{then} \\ & \quad b_{1,1,1,1}\, \text{else}\, b_{1,1,1,0})\,\text{in} \\ & ... \\ & \text{let}\, x_{N_{B^1},N_C,N_C} = \text{random}_{0.5}(\lambda x : \text{bool}\,.\,\text{if}\, x \\ & \quad \text{then}\, b_{N_{B^2},N_C,N_C,1}\, \text{else}\, b_{N_{B^2},N_C,N_C,0})\,\text{in} \\ & (x_{1,1}, ..., x_{N_{B^1},N_C}, x_{1,1,1}, ..., x_{N_{B^2},N_C,N_C}), \end{aligned} \tag{4}$$

representing an expression which uniformly samples a possible world. The term $Z_Q$ in Eq. 3 is constructed as a list of pairs $[(t, \tau_t), t \in \text{Types}(T)]$, where $T$ is defined in Supp. Eq. 14 (representing tuples for all combinations of assumptions and possible words).

For each $t \in \text{Types}(T)$, we consider the context $\Gamma_t = \Gamma', x : t$, and construct the sets $R_{Q,t}$ of expressions $r$ such that $\Gamma_t \vdash r : Q$, $S_{Q,t}$ of expressions $s$ such that $\Gamma_t \vdash s : Q \rightarrow \perp$, and $K_{Q,t}$ of expressions $k$ such that $\Gamma_t \vdash k : \perp$. If $K \neq \emptyset$, we set $\tau_t = k$ for an arbitrary $k \in K$. Otherwise, we must have either $R = \emptyset$ or $S = \emptyset$ (since if neither is the case $(sr) \in K$ for arbitrary $s \in S$, $r \in R$, and the term $D$ in Eq. 4 ensures at least one is non-empty). If $R = \emptyset \wedge K = \emptyset$, we set $\tau_t = s$ for an arbitrary $s \in S$, and if $S = \emptyset \wedge K = \emptyset$, we set $\tau_t = r$ for an arbitrary $r \in R$. $(\lambda x.Z_Q)$ thus returns a proof of $Q$, $\neg Q$, or $\perp$, depending on which of these can be constructed given the type of the input argument passed. We define the probability of a query in terms of the probability that $q_Q$ reduces by $\beta\rho$-reduction to a term $\tau$ of type $Q$ given that it does not reduce to an inconsistency (where $\tau \in \text{NF}_{\lambda P^{\Sigma\,\text{bool}}}$):

$$P(Q|\Gamma_0, \Gamma_1, P) \;\;=\;\; \frac{P(q_Q \twoheadrightarrow^{\rho}_{\beta} \tau : Q)}{1 - P(q_Q \twoheadrightarrow^{\rho}_{\beta} \tau : \perp)}. \tag{5}$$

The probability in Eq. 5 can be evaluated by sampling repeated reductions of Eq. 3, and rejecting those samples returning $\tau : \perp$. Writing $x_W$ for a type $x_W \in \text{Types}(D)$, i.e. a possible world, and $\mathcal{P}(.)$ for the powerset operator, we can show:

**Theorem 2.** *(DTN distribution over possible worlds): A DTN defined as above represents the following distribution over possible worlds:*

$$\begin{aligned} P(x_W|\Gamma_{0,1}, P) \propto \textstyle\sum_{H \in \mathcal{P}(\{1...N_F\})} (\prod_{j \in H} p_j) \cdot \\ (\textstyle\prod_{j \notin H}(1 - p_j))[\forall j \in H, \exists y\, \text{s.t.}\, \Gamma', t : x_W \vdash y : F_j]. \end{aligned} \tag{6}$$

A proof is provided in the supplement (Supp. Prop. 8).

## 4 Equivalence with Markov Logic Networks

As defined in [4], a Markov Logic Network (MLN) $L$ is a set of pairs $(F_i, w_i)$, where $F_i$ is a formula in first order logic and $w_i$ is an arbitrary real number. $L$ acts as a template, which when combined with a finite set of constants $C = \{c_1, c_2, ..., c_{|C|}\}$ (which includes and may extend the constants used to build the $F_i$'s, defining the domain over which the model ranges) constructs a *ground Markov Network*, $M_{L,C}$. $M_{L,C}$ contains a binary node for every possible grounding of the predicates in $L$ (a *ground predicate* being a formula involving only the applications of a predicate to a tuple of constants). The network also contains cliques for all possible groundings of each formula $F$ in $L$, where the clique derived from the $j$'th grounding of formula $i$ is associated with a clique potential $\psi_{i,j}$ which is 1 if the ground formula is not satisfied, and $e^{w_i}$ if it is. A possible world is fixed by a joint setting of the nodes of $M_{L,C}$, which we denote by $\mathbf{x} \in \{0,1\}^P$, where $P$ is the number of ground predicates (nodes) in the network. $M_{L,C}$ thus defines a distribution over possible worlds, which can be expressed as a random variable $X$ which ranges over the settings $\mathbf{x}$:

$$P(X = \mathbf{x}) = \frac{1}{Z} \prod_{ij} \psi_{ij}(\mathbf{x}_{ij}) = \frac{1}{Z} \prod_{i} (e^{w_i})^{n_i(\mathbf{x})}, \quad (7)$$

where $Z$ is the partition function $Z = \sum_{\mathbf{x}} \prod_{ij} \psi_{ij}(\mathbf{x}_{ij})$, and $n_i(\mathbf{x})$ denotes the number of true groundings of formula $F_i$ in joint setting $\mathbf{x}$. The probability that a given formula holds in a possible world distributed according to Eq. 7 can be written $P(F|L,C) = \sum_{\mathbf{x} \in \mathcal{X}_F} P(\mathbf{x}|L,C)$, where $\mathcal{X}_F$ is the subset of possible worlds in which query formula $F$ is true.

To draw a relation with DTNs, we require that an MLN is expressed in a canonical form, using only non-positive weights $w_i$. This can always be achieved as follows: For an arbitrary MLN $L$, we form an equivalent network $L'$, where, for pair $(F_i, w_i) \in L$ we set $(F'_i, w'_i) \in L'$ with $F'_i = F_i$, $w' = w$ if $w \leq 0$, and $F'_i = \neg F_i$, $w' = -w$ otherwise. This transformation will preserve the distribution across worlds, since the ratio $\psi_i(\mathbf{x}_1)/\psi_i(\mathbf{x}_2)$ for a world which satisfies $F_i$, $\mathbf{x}_1$, and one which does not, $\mathbf{x}_2$, is $e^w$ in both networks $M_{L,C}$ and $M_{L',C}$.

We can then demonstrate the following equivalence:

**Theorem 3.** *(Equivalence of DTNs and MLNs): We have the following canonical maps, which preserve distributions over possible worlds:*

*(a) Given a ground Markov Logic Network $M_{L,C}$, $L = (F'_i, w'_i \leq 0)$, we map it to a DTN $\{\Gamma_0, \Gamma_1, P\}$, $\Gamma_1 = \Gamma_0, < f_i : F''_i, i = 1 : N_F >$, such that:*

$$\begin{aligned} F''_i &= \hat{F}'_i \to \bot \\ p_i &= (1 - \exp(w'_i)), \end{aligned} \quad (8)$$

*where $\hat{F}$ denotes the type corresponding to formula $F$ under the Curry-Howard correspondence,*

*(b) Given a DTN $\{\Gamma_0, \Gamma_1, P\}$, $\Gamma_1 = \Gamma_0, < f_i : F_i, i = 1 : N_F >$, we map it to an MLN $M_{L,C}$, $L = (F'_i, w_i)$, such that:*

$$\begin{aligned} F'_i &= \neg \hat{\hat{F}}_i \\ w_i &= \log(1 - p_i), \end{aligned} \quad (9)$$

*where $\hat{\hat{F}}$ denotes the formula corresponding to type $F$ under the Curry-Howard correspondence.*

A proof is provided in the supplement (Supp. Prop. 1).

A direct corollary of this equivalence is therefore:

**Corollary 1.** *DTNs are a universal representation for distributions across finitely many discrete variables.*

*Proof.* As shown in [4], MLNs are a universal representation for finite discrete distributions.

## 5 Discussion

We have used a restricted system here to show that stochastic functions can be embedded in dependent type theory (DTT), and how this naturally leads to a probabilistic logic equivalent to Markov Logic Networks. In general, we would wish to make use of the full resources of DTT [10, 11], such as inductive and recursive definitions, identity types and higher inductive types to formulate more general PDTS's, in which richer probabilistic logics may be embedded. We note however that the current framework is suggestive of a number of directions for investigation both with respect to the specific models investigated here and more generally. For instance, the relationship between DTNs and MLNs suggests that automatic theorem proving techniques may be generally applied in the context of inference in graphical models (which may naturally be expressed as MLNs), since Eq. 5 may be evaluated by a meta-algorithm which repeatedly samples assumptions and searches for proofs of $Q$, $\neg Q$ or $\bot$. The recent application of deep-learning techniques to automated and interactive theorem proving [12, 13] is potentially a powerful tool in this context.

Another suggestive direction is model learning. Recent approaches have shown that deep-learning methods can be fully integrated with probabilistic programming [14]. Our PDTS framework is naturally treated as a probabilistic programming framework, and we can straightforwardly incorporate deep neural networks as stochastic or deterministic functions within the PDTS outlined. In the logical context, this corresponds to learning formulae of arbitrary complexity in a DTN. Additionally, the ambient PDTS of our framework can naturally represent both models and inference procedures applied to models as stochastic programs, as in [14].

Finally, we note that fully integrating dependent types in probabilistic programming languages has the potential to draw a tight connection between machine learning techniques and recent computational approaches in the foundations of mathematics, such as UniMath [15].

## References

[1] N.D. Goodman, V.K. Mansinghka, D. Roy, K. Bonawitz, and J.B. Tenenbaum. Church: A language for generative models. *UAI*, 2008.

[2] J. Borgström, A.D. Gordon, M. Greenberg, J. Margetson, and J. Van Gael. Measure Transformer Semantics for Bayesian Machine Learning. In *Programming Languages and Systems*, 77-96, Springer Berlin Heidelberg, 2011.

[3] P. Martin Löf. An Intuitionistic Theory of Types: Predicative Part. In *Logic Colloquim, 1973*, North-Holland, 1975.

[4] M. Richardson, and P. Domingos. Markov Logic Networks. *Machine Learning*, 62(1-2):107-136, 2006.

[5] K.S. Ng, and J.W. Lloyd. Probabilistic Reasoning in a Classical Logic. *Journal of Applied Logic*, 7(2):218-238, 2007.

[6] M. Hutter, J.W. Lloyd, K.S. Ng, and W.T.B. Uther. Probabilities on Sentences in an Expressive Logic. *Journal of Applied Logic*, 11(4):386-420, 2013.

[7] R. Cooper, S. Dobnik, S. Lappin, and S. Larsson. A Probabilistic Rich Type Theory for Semantic Interpretation. *Proceedings of the EACL 2014 Workshop on Type Theory and Natural Language Semantics*, 72-79, 2014.

[8] H. P. Barendregt. Lambda Calculi with Types. In *Handbook of Logic in Computer Science*, Vol 2:1-189, 1992.

[9] S. Thompson. *Type Theory and Functional Programming*, Addison-Wesley, 1991.

[10] P. Martin-Löf. ”Constructive mathematics and computer programming.” In *Studies in Logic and the Foundations of Mathematics*, vol. 104, pp. 153-175. Elsevier, 1982.

[11] The *Univalent Foundations Program*, *Homotopy Type Theory: Univalent Foundations of Mathematics*, Princeton NJ, Institute for Advanced Study, 2013, https://homotopytypetheory.org/book.

[12] C. Kaliszyk, F. Chollet, and C. Szegedy. Holstep: A machine learning dataset for higher-order logic theorem proving, *ICLR*, 2017.

[13] T. Rocktäschel, and S. Riedel. End-to-end differentiable proving. *NIPS* 2017.

[14] D. Tran, M. D. Hoffman, R. A. Saurous, E. Brevdo, K. Murphy, and D. M. Blei. Deep probabilistic programming, *ICLR*, 2017.

[15] V. Voevodsky, B. Ahrens, and D. Grayson, *UniMath*, 2014, https://github.com/UniMath.

# Dependent Type Networks: A Probabilistic Logic via the Curry-Howard Correspondence in a System of Probabilistic Dependent Types Supplementary Materials

Jonathan Warrell[12*], Mark Gerstein[123]

[1] *Program in Computational Biology and Bioinformatics, Yale University, New Haven, CT 06520, USA*
[2] *Department of Molecular Biophysics and Biochemistry, Yale University, New Haven, CT 06520, USA*
[3] *Department of Computer Science, Yale University, New Haven, CT 06520, USA*

*jonathan.warrell{@yale.edu, @gmail.com}

**Abstract.** We first introduce a *Probabilistic Dependent Type System* (PDTS) via a functional language based on a subsystem of intuitionistic type theory including dependent sums and products, which is expanded to include stochastic functions. We provide a sampling-based semantics for the language based on non-deterministic beta reduction. Further, we formulate a probabilistic logic by defining a *Dependent Type Network* (DTN) within the PDTS, where the DTN is derived as a direct result of the Curry-Howard correspondence. We show that DTNs are equivalent to Markov Logic Networks, and hence are a universal representation for discrete distributions. These supplementary materials provide an extended presentation of the main paper.

## 1 Introduction

Stochastic functional programming languages have used non-deterministic beta reduction to provide a probabilistic semantics for languages based on the untyped lambda calculus [1] and probabilistic type systems based on the simply-typed lambda calculus [2]. Here, we show that a similar approach can be used to provide semantics for a probabilistic type system based on intuitionistic type theory [3], hence including also dependent types. We call these *Probabilistic Dependent Type Systems* (PDTS).

We demonstrate that a probabilistic logic can be naturally formulated in our PDTS framework, which we term a *Dependent Type Network* (DTN), using the Curry-Howard correspondence. The logic we derive is distinct from previous formulations of probabilistic logic: Previous approaches have involved associating weights with sentences which induce a distribution over interpretations (possible worlds) [4]; formulating conditions for consistently assigning probabilities to sentences or interpretations in a simply-typed higher-order language [5, 6]; or assigning terms and deterministic functions probabilistically to types via probabilistic type judgments [7]. In contrast, we formulate an intuitionistic type system of non-deterministic functions, and derive a probabilistic logic as a subsystem. We show that DTNs are equivalent to Markov Logic Networks (MLNs) [4], and hence are a universal representation for discrete distributions.

Sec. 2 outlines a deterministic dependent type system, which is then generalized to include non-deterministic functions in Sec. 3, hence forming a PDTS. Sec. 4 then uses the PDTS to derive DTNs as a probabilistic logic via the Curry-Howard correspondence, and shows that a canonical translation can be made between DTNs and MLNs.

## 2 A Dependent Type System: $\lambda P^{\Sigma\,\mathrm{bool}}$

We give an initial overview of a system we will call $\lambda P^{\Sigma\,\mathrm{bool}}$. This will be a typed lambda calculus with dependent products (hence based on the system $\lambda P$, see [8]), dependent sums ($\Sigma$-types), and $\mathrm{bool}$ and $\mathrm{unit}$ base types.

The language includes the following sets of base constants $\mathcal{C}_0$, and sorts, denoted $\mathcal{S}$:

$$\begin{aligned} \mathcal{C}_0 &::= 1 \ \mid \ \mathrm{true} \ \mid \ \mathrm{false} \ \mid \ \mathrm{unit} \ \mid \ \mathrm{bool} \\ \mathcal{S} &::= * \ \mid \ \square \end{aligned} \tag{1}$$

We denote the constants collectively as $\mathcal{C} = \mathcal{C}_0 \cup \mathcal{S}$. We also have a countably infinite supply of variables, $\mathcal{V} = \{a, b, c, ...A, B, C...a_1, b_1, c_1...a_2...\}$. An abstract syntax (see [8]) for *values* $\mathcal{U}$ and *pseudo-expressions* $\mathcal{T}$ in our language can be specified as follows:

$$\begin{aligned} \mathcal{U} &::= \mathcal{C} \ \mid \ \mathcal{V} \ \mid \ (\mathcal{U},\mathcal{U})_{\mathcal{T}} \\ \mathcal{T} &::= \mathcal{U} \ \mid \ (\mathcal{T}\mathcal{T}) \ \mid \ \lambda\mathcal{V} : \mathcal{T}.\mathcal{T} \ \mid \ \mathrm{if}\,\mathcal{V}\,\mathrm{then}\,\mathcal{T}\,\mathrm{else}\,\mathcal{T} \ \mid \\ &\quad \pi_1\mathcal{T} \ \mid \ \pi_2\mathcal{T} \ \mid \ \textstyle\sum\mathcal{V} : \mathcal{T}.\mathcal{T} \ \mid \ \textstyle\prod\mathcal{V} : \mathcal{T}.\mathcal{T} \end{aligned} \tag{2}$$

We define inductively the *free variables* of a pseudo-expression $FV(\tau)$, such that $FV(c) = \emptyset$ for $c \in \mathcal{C}$, $FV(v) = \{v\}$ for $v \in \mathcal{V}$, $FV((u_1, u_2)) = FV(u_1) \cup FV(u_2)$ for $u_1, u_2 \in \mathcal{U}$, $FV(\tau_1\tau_2) = FV(\tau_1) \cup FV(\tau_2)$ for $\tau_1, \tau_2 \in \mathcal{T}$, $FV(\pi_1\tau) = FV(\pi_2\tau) = FV(\tau)$, $FV(\text{if } v \text{ then } \tau_1 \text{ else } \tau_2) = FV(\tau_1) \cup FV(\tau_2) \cup FV(v)$, and $FV(\lambda v : \tau_1.\tau_2) = FV(\sum v : \tau_1.\tau_2) = FV(\prod v : \tau_1.\tau_2) = FV(\tau_1) \cup FV(\tau_2) - \{v\}$. Any variable appearing in $\tau$ which is not a free variable is bound. Pseudo-expressions which can be converted into one another by renaming variables such that no free variable becomes bound after renaming are identified as equivalent (by alpha conversion), and we write $\tau_1 \equiv \tau_2$ when this is the case. Further, the notion of beta reduction between pseudo-expressions, denoted $\tau_1 \rightarrow_\beta \tau_2$, is specified by the following rules:

$$\begin{aligned}
\pi_1(u_1, u_2)_\tau &\rightarrow_\beta u_1 \\
\pi_2(u_1, u_2)_\tau &\rightarrow_\beta u_2 \\
\text{if true then } \tau_1 \text{ else } \tau_2 &\rightarrow_\beta \tau_1 \\
\text{if false then } \tau_1 \text{ else } \tau_2 &\rightarrow_\beta \tau_2 \\
(\lambda x : \tau_1.\tau_2)\tau_3 &\rightarrow_\beta \tau_2[x := \tau_3] \quad \text{(see conditions below)},
\end{aligned} \tag{3}$$

where $u_1, u_2 \in \mathcal{U}$, $\tau_1, \tau_2, \tau_3, \tau \in \mathcal{T}$, $b_1, b_2 \in \{\text{true}, \text{false}\}$, and $\tau_2[x := \tau_3]$ denotes the result of substituting $\tau_3$ for $x$ in $\tau_2$. For rule 5, we also impose the conditions that no variable in $FV(\tau_3)$ becomes bound on substitution (transforming $\tau_2$ by alpha conversion if necessary before substitution to prevent this), and that $\tau_3 \in \mathcal{U}$ if $\tau_2$ contains any sub-expressions of the form $(x, y)$, $(y, x)$ or $(x, x)$. In addition to the rules above, we extend the beta reduction relation as follows: letting $\tau_2$ be a *free sub-(pseudo)-expression* of $\tau_1$, meaning that $\tau_1, \tau_2 \in \mathcal{T}$, $\tau_2$ is a substring of $\tau_1$, and no free variable of $\tau_2$ is bound in $\tau_1$ (by a $\lambda, \prod$ or $\sum$ construct), we have that $\tau_1 \rightarrow_\beta \tau_1'$ whenever $\tau_2 \rightarrow_\beta \tau_2'$ by one of the above rules (Eq. 3), and $\tau_1'$ is the result of replacing $\tau_2$ for $\tau_2'$ in $\tau_1$. Further, we write $\tau_1 =_\beta \tau_2$ iff $\tau_1 \twoheadrightarrow_\beta \tau_2$ or $\tau_2 \twoheadrightarrow_\beta \tau_1$, where $\twoheadrightarrow_\beta$ is the reflexive transitive closure of $\rightarrow_\beta$.

The pseudo-expressions for which we can infer types will be the expressions proper (legal expressions/terms) of the language. For this purpose, we define a *statement* to be of the form $\tau_0 : \tau_1$ (pronounced $\tau_0$ is of type $\tau_1$). A *context* is a finite linearly ordered set of statements, $\Gamma = <\tau_{1,0} : \tau_{1,1}, \tau_{2,0} : \tau_{2,1}, ..., \tau_{N,0} : \tau_{N,1}>$, where $\tau_{n,0} \in \mathcal{V}$, $n = 1...N$, and for $n_1 \neq n_2$, $\tau_{n_1,0} \neq \tau_{n_2,0}$. We write $<>$ for the empty context, and $\Gamma, x : \tau$ for the result of appending $x : \tau$ to $\Gamma$. We can then specify rules for type inference as follows:

$$\text{(axioms):} \quad <> \vdash * : \square, \quad \text{bool} : *, \quad \text{unit} : *, \quad 1 : \text{unit}, \quad \text{true} : \text{bool}, \quad \text{false} : \text{bool},$$

$$\text{(start):} \quad \frac{\Gamma \vdash A : s}{\Gamma, x : A \vdash x : A} \; x \notin \Gamma$$

$$\text{(weakening):} \quad \frac{\Gamma \vdash A : B \qquad \Gamma \vdash C : s}{\Gamma, x : C \vdash A : B} \; x \notin \Gamma$$

$$\text{(type/kind formation):} \quad \frac{\Gamma \vdash A : * \qquad \Gamma, x : A \vdash B : s}{\Gamma \vdash (\prod x : A.B) : s, \;\; (\sum x : A.B) : s}$$

$$\text{(application):} \quad \frac{\Gamma \vdash F : (\prod x : A.B) \qquad \Gamma \vdash a : A}{\Gamma \vdash Fa : B[x := a]}$$

$$\text{(abstraction):} \quad \frac{\Gamma, x : A \vdash b : B \qquad \Gamma \vdash (\prod x : A.B) : s}{\Gamma \vdash (\lambda x : A.b) : (\prod x : A.B)}$$

$$\text{(if):} \quad \frac{\Gamma \vdash a : \text{bool} \qquad \Gamma \vdash b_1 : B[x := \text{true}] \qquad \Gamma \vdash b_2 : B[x := \text{false}]}{\Gamma \vdash (\text{if } a \text{ then } b_1 \text{ else } b_2) : B[x := a]}$$

$$\text{(products (1)):} \quad \frac{\Gamma, x : A \vdash b : B \qquad \Gamma \vdash (\sum x : A.B) : s}{\Gamma \vdash (x, b)_{(\sum x:A.B)} : (\sum x : A.B)} \; x, b \in \mathcal{U}$$

$$\text{(products (2)):} \quad \frac{\Gamma \vdash c : (\sum x : A.B)}{\Gamma \vdash (\pi_1 c) : A, \;\; (\pi_2 c) : (B[x := (\pi_1 c)])}$$

$$\text{(type conversion):} \quad \frac{\Gamma \vdash A : B \qquad \Gamma \vdash C : s \qquad B =_\beta C}{\Gamma \vdash A : C}, \tag{4}$$

where $s$ ranges over $\mathcal{S}$ and $\tau_1 \rightarrow \tau_2$ is shorthand for $\prod x : \tau_1.\tau_2$ in the case that $\tau_2$ is independent of $x$. In the following, we will also use the notation $\tau_1 \times \tau_2$ for $\sum x : \tau_1.\tau_2$ where $\tau_2$ is independent of $x$, $(\text{let}\, x = \tau_1 \,\text{in}\, \tau_2)$ for $(\lambda x : A.\tau_2)\tau_1$, where $\tau_1 : A$, and $(x, y, z)$ for $(x, (y, z))$ and similarly for longer tuples, where we omit tuple type-tags for clarity.

The system $\lambda P^{\Sigma\,\text{bool}}$ forms a subsystem of the most general system considered in [9] (TT), and as such can be shown to be strongly normalizing (see Theorem 5.36, [9]), implying that $\beta$ reduction sequences of all valid expressions can always be extended to terminate in a normal form (an expression which cannot be reduced further), and that each expression has a unique normal form up to alpha equivalence. We can thus write $\text{Norm}(\tau)$ for the unique normal form of $\tau$, where $\tau$ is a typable expression. Further properties which can be demonstrated in $\lambda P^{\Sigma\,\text{bool}}$ include *preservation* ($\beta$-reduction preserves types up to beta equivalence, in that for a reduction step $\tau_1 \rightarrow_\beta \tau_2$, where $\tau_1 : \tau_3$ and $\tau_2 : \tau_4$, we have that $\tau_3 =_\beta \tau_4$), and *progress* (any expression not in normal form has a valid $\beta$ reduction). The second of these properties follows directly from strong normalization, and we give a proof of the first in the Appendix (Prop. 2).

## 3 Probabilistic Dependent Types: $\lambda P\rho^{\Sigma\,\text{bool}}$

We now enlarge the $\lambda P^{\Sigma\,\text{bool}}$ system to include primitives for sampling from distributions, using a similar syntax to [2]. The calculus developed in [2] is based on the simple typed lambda calculus, and includes only distributions over values within a type. Since our calculus also includes dependent types, we naturally derive expressions which represent distributions over terms of multiple types, distributions over types, as well as functions whose return type is probabilistically determined. The language we develop in this section will be called $\lambda P\rho^{\Sigma\,\text{bool}}$, and contains as a subsystem the fragment of the Fun language in [2] only involving Bernoulli distributions (and without observe statements)[4] when the dependent types are removed.

We begin by expanding the syntax of Sec. 2:

$$\begin{aligned}
&\mathcal{U} ::= \mathcal{C}_0 \mid \mathcal{V} \mid (\mathcal{U},\mathcal{U})_\mathcal{T} \\
&\mathcal{Z} ::= \{\} \mid \{(\mathcal{T},\mathcal{T}),\ \mathcal{Z}\} \\
&\mathcal{T} ::= \mathcal{U} \mid (\mathcal{T}\mathcal{T}) \mid \lambda\mathcal{V} : \mathcal{T}.\mathcal{T} \mid \text{if}\,\mathcal{V}\,\text{then}\,\mathcal{T}\,\text{else}\,\mathcal{T} \mid \\
&\qquad \pi_1\mathcal{T} \mid \pi_2\mathcal{T} \mid \textstyle\sum \mathcal{V} : \mathcal{T}.\mathcal{T} \mid \prod \mathcal{V} : \mathcal{T}.\mathcal{T} \mid (\lambda\mathcal{V}.\mathcal{Z})\mathcal{T} \mid \text{random}_\rho(\mathcal{T})
\end{aligned} \tag{5}$$

where $\rho$ is any real number in the interval $(0\ 1)$, and the new syntactic construction $\mathcal{Z}$ allows us to form lists/ordered sets of paired terms. Since the syntax in Eq. 5 is an expansion of that in Eq. 2, all pseudo-expressions of $\lambda P^{\Sigma\,\text{bool}}$ will also be pseudo-expressions of $\lambda P\rho^{\Sigma\,\text{bool}}$. We denote the set of common pseudo-expressions as $\mathcal{T}_{\lambda P^{\Sigma\,\text{bool}}}$. Next, we define inductively a weighted reduction relationship ($\beta\rho$-reduction) on pseudo-expressions, $\tau_1 \rightarrow^\rho_\beta \tau_2$ via:

$$\begin{aligned}
\pi_1(u_1, u_2)_\mathcal{T} &\rightarrow^1_\beta \quad u_1 \\
\pi_2(u_1, u_2)_\mathcal{T} &\rightarrow^1_\beta \quad u_2 \\
\text{if true then}\,\tau_1\,\text{else}\,\tau_2 &\rightarrow^1_\beta \quad \tau_1 \\
\text{if false then}\,\tau_1\,\text{else}\,\tau_2 &\rightarrow^1_\beta \quad \tau_2 \\
\text{random}_\rho(\tau) &\rightarrow^\rho_\beta \quad (\tau\,\text{true}) \\
\text{random}_\rho(\tau) &\rightarrow^{(1-\rho)}_\beta (\tau\,\text{false}) \\
(\lambda x : \tau_1.\tau_2)\tau_3 &\rightarrow^1_\beta \quad \tau_2[x := \tau_3] \quad \text{if } \tau_3 \in \mathcal{T}_{\lambda P^{\Sigma\,\text{bool}}}, \text{and conditions as in Eq. 3 rule 5,} \\
(\lambda x.Z)\tau_3 &\rightarrow^1_\beta \quad (\lambda x : \tau_1.\tau_2)\tau_3 \quad \text{if } \tau_3 \in \mathcal{T}_{\lambda P^{\Sigma\,\text{bool}}}, \tau_3 : \tau_1, (\tau_1, \tau_2)\,\text{in}\,Z, \\
&\qquad\qquad \text{and } \forall \tau_2' \neq \tau_2, \neg((\tau_1, \tau_2')\,\text{in}\,Z).
\end{aligned} \tag{6}$$

[4] The observe statement in [2] allows a distribution to be conditioned on (and updated by) evidence. As we discuss in Sec. 4.1, such conditioning can be simulated by adding the conditioning formula with a weight tending to $\infty$ in the context of Markov Logic Networks, and hence also in Dependent Type Networks by the correspondence we describe in Sec. 4.3. Further, in a general PDTS we could say return the unit value from a probabilistic function if the conditional is not satisfied. We thus simplify our analysis by not including the statement in our syntax.

where, as before, $u_1, u_2 \in \mathcal{U}$, $\tau_1, \tau_2, \tau_3, \tau \in \mathcal{T}$, $b_1, b_2 \in \{\text{true}, \text{false}\}$, and here $Z \in \mathcal{Z}$, while we write in for the transitive closure of the set membership relation where $Z$ is interpreted as denoting a set (alternatively, in denotes list membership if $Z$ is viewed as a list data structure). We extend the reduction relation by allowing $\tau_1 \rightarrow^{\rho}_{\beta} \tau'_1$ whenever $\tau_2 \rightarrow^{\rho}_{\beta} \tau'_2$ by one of the above rules, and $\tau'_1$ is the result of replacing $\tau_2$ for $\tau'_2$ in $\tau_1$, where $\tau_2$ is either (i) the leftmost-outermost reducible sub-(psuedo)-expression of the form $\tau_2 = \text{random}_{\rho}(\tau_3)$ or $\tau_2 = (\lambda x.Z)\tau_3$, or (ii) if no reducible sub-(psuedo)-expression of the forms in (i) exist, the leftmost-outermost free sub-(pseudo)-expression of $\tau_1$ as in Sec. 2. Here, *leftmost-outermost* refers to the sub-expression found by searching the parse tree of a (pseudo)-expression top-down, traversing the left subtree of a non-reducible expression before the right (see [9], Def. 2.13). These conditions ensure that there is at most one sub-(pseudo)-expression whose reduction is allowed at any point. The relationship $\tau_1 \rightarrow^{\rho}_{\beta} \tau_2$ can be given the operational semantics '$\tau_1$ reduces to $\tau_2$ with probability $\rho$'. We can further define the relation $\tau_1 \twoheadrightarrow_{\beta} \tau_2$ to hold when there exists any reduction chain between $\tau_1$ and $\tau_2$ such that no step has zero probability, and $\tau_1 \twoheadrightarrow^{\rho_0}_{\beta} \tau_2$ to hold when the sum of the products of the $\rho$ values across all such chains is $\rho_0$ (where $\rho_0 \leq 1$, see Appendix Prop. 5). Further, we note that the rules for reduction of $\lambda$ terms in Eq. 6 whereby a term may not be substituted until it is reduced to a term in $\mathcal{T}_{\lambda P^{\Sigma\,\text{bool}}}$ enforce a 'stochastic memoization' property on reduction sequences as defined in [1] (a random statement will not be duplicated by substitution, and reduced in alternative ways in a single reduction sequence).

In [2], type assignment rules are given which consistently assign a single type to all expressions such that a *preservation* property holds: in any probabilistic reduction sequence, the types of all terms are preserved at each reduction step (up to $\beta$ equivalence). Since our language contains dependent types, such a property cannot be expected to hold, as a given expression may reduce to expressions with different types by different reduction sequences. Instead of assigning types to all expressions in $\lambda P\rho^{\Sigma\,\text{bool}}$, only expressions already typed in $\lambda P^{\Sigma\,\text{bool}}$ by the rules of Eq. 4 will be typed directly. To identify general valid expressions in $\lambda P\rho^{\Sigma\,\text{bool}}$, we define two operators Types and Reductions on pseudo-expressions, where $\text{Reductions}(\tau)$ will return the set of normal forms to which a given psuedo-expression reduces with non-zero probability, and $\text{Types}(\tau)$ will return the set of types corresponding to those normal forms (we will ensure all reduction sequences of legal expressions in $\lambda P\rho^{\Sigma\,\text{bool}}$ end in $\lambda P^{\Sigma\,\text{bool}}$ normal forms). We define these operators inductively. For each pseudo-expression $\tau_1$ of $\lambda P^{\Sigma\,\text{bool}}$, we set $\text{Types}(\tau_1) = \{\tau_2\}$ when $\tau_1 : \tau_2$, $\tau_2 \in \text{NF}_{\lambda P^{\Sigma\,\text{bool}}}$, and $\text{Types}(\tau_1) = \{\text{no-type}\}$ otherwise. Further, $\text{Reductions}(\tau_1) = \{\text{Norm}(\tau_1)\}$ in the former case, and $\emptyset$ in the latter. The following inductive rules are then used to define these operators for all valid expressions in $\lambda P\rho^{\Sigma\,\text{bool}}$ (where we let $\mathcal{T}_x$ be the set of $\lambda P\rho^{\Sigma\,\text{bool}}$ pseudo-expressions containing no reducible sub-expressions containing the variable $x$, including sub-expressions within $\mathcal{Z}$ constructions):

$$\text{(random):}\quad \frac{\begin{array}{c}\Gamma \vdash \text{Reductions}(F) = \{\lambda x : \text{bool}\,.a_n, n = 1...N_1\},\\ \Gamma \vdash \text{Types}(F) = \{\prod x : \text{bool}\,.A_n, n = 1...N_2\},\end{array}}{\begin{array}{c}\Gamma \vdash \text{Reductions}(\text{random}_{\rho}(F)) = \bigcup_{b\in\{\text{true},\text{false}\}}\{\text{Norm}(a_n[x := b]), n = 1...N_1\}\\ \Gamma \vdash \text{Types}(\text{random}_{\rho}(F)) = \bigcup_{b\in\{\text{true},\text{false}\}}\{\text{Norm}(A_n[x := b]), n = 1...N_2\}\end{array}}$$

$$\text{(if):}\quad \frac{\begin{array}{c}\Gamma \vdash a : \text{bool}, \text{Reductions}(a_1) = \{b_{1,n}, n = 1...N_{1,1}\},\\ \text{Reductions}(a_2) = \{b_{2,n}, n = 1...N_{2,1}\},\\ \text{Types}(a_1) = \{B_{1,n}, n = 1...N_{1,2}\},\\ \text{Types}(a_2) = \{B_{2,n}, n = 1...N_{2,2}\}\end{array}}{\begin{array}{c}\Gamma \vdash \text{Reductions}(\text{if}\, a \,\text{then}\, a_1 \,\text{else}\, a_2) = \{\text{if}\, a \,\text{then}\, b_{1,n_1} \,\text{else}\, b_{2,n_2}, n_1 = 1...N_{1,1}, n_2 = 1...N_{2,1}\}\\ \Gamma \vdash \text{Types}(\text{if}\, a \,\text{then}\, a_1 \,\text{else}\, a_2) = \{\text{if}\, a \,\text{then}\, B_{1,n_1} \,\text{else}\, B_{2,n_2}, n_1 = 1...N_{1,2}, n_2 = 1...N_{2,2}\}\end{array}}$$

$$\text{(abstraction):}\quad \frac{\begin{array}{c}\Gamma, x : A \vdash \text{Reductions}(b) = \{b_n, n = 1...N_1\},\\ \Gamma, x : A \vdash \text{Types}(b) = \{B_n, n = 1...N_2\}, \quad b \in \mathcal{T}_x\end{array}}{\begin{array}{c}\Gamma \vdash \text{Reductions}(\lambda x : A.b) = \{\lambda x : A.b_n, n = 1...N_1\}\\ \Gamma \vdash \text{Types}(\lambda x : A.b) = \{\prod x : A.B_n, n = 1...N_2\}\end{array}}$$

$$\text{(application (1)):}\quad \frac{\begin{array}{c}\Gamma \vdash \text{Reductions}(a) = \mathcal{A}, \text{Reductions}(F) = \{\lambda x : A.b_n, n = 1...N_1\},\\ \Gamma \vdash \text{Types}(a) = \{A\}, \text{Types}(F) = \{\prod x : A.B_n, n = 1...N_2\}\end{array}}{\begin{array}{c}\Gamma \vdash \text{Reductions}(Fa) = \bigcup_{a'\in\mathcal{A}}\{\text{Norm}(b_n[x := a']), n = 1...N_1\}\\ \Gamma \vdash \text{Types}(Fa) = \bigcup_{a'\in\mathcal{A}}\{\text{Norm}(B_n[x := a']), n = 1...N_2\}\end{array}}$$

$$\text{(application (2)):} \quad \frac{\begin{array}{c}\Gamma \vdash \text{Reductions}(a) = \mathcal{A}, \text{Types}(a) = \mathcal{A}', \\ \forall A \in \mathcal{A}', (A : *) \wedge (\exists (A, b) \in\in Z, \text{s.t.} \forall b' \neq b, \neg((A, b') \in\in Z) \wedge \\ (\forall a' \in \mathcal{A} \,\text{s.t.}\, a' : A, \forall B \in \text{Types}(b[x := a']), B : s) \wedge (b \in \mathcal{T}_x))\end{array}}{\begin{array}{c}\Gamma \vdash \text{Reductions}((\lambda x.Z)a) = \bigcup_{a' \in \mathcal{A}, (A,b) \in\in Z, a' : A} \text{Reductions}(b[x := a']) \\ \Gamma \vdash \text{Types}((\lambda x.Z)a) = \bigcup_{a' \in \mathcal{A}, (A,b) \in\in Z, a' : A} \text{Types}(b[x := a'])\end{array}}$$

$$(\textstyle\prod/\sum \text{ formation):} \quad \frac{\begin{array}{c}\Gamma \vdash A : *, \quad \Gamma, x : A \vdash \text{Reductions}(B) = \{B_n, n = 1...N_1\}, \\ \Gamma, a : A \vdash \text{Types}(B) = \{s\}, \quad s \in \{*, \Box\}, \quad B \in \mathcal{T}_x\end{array}}{\begin{array}{c}\Gamma \vdash \text{Reductions}(\prod x : A.B) = \{\prod x : A.B_n, n = 1...N_1\} \\ \Gamma \vdash \text{Reductions}(\sum x : A.B) = \{\sum x : A.B_n, n = 1...N_1\} \\ \Gamma \vdash \text{Types}(\prod x : A.B) = \{s\} \\ \Gamma \vdash \text{Types}(\sum x : A.B) = \{s\}\end{array}}$$

$$\text{(products):} \quad \frac{\Gamma \vdash \text{Reductions}(a) = \mathcal{A}, \text{Types}(a) = \{\prod a : A.B\}}{\begin{array}{c}\Gamma \vdash \text{Reductions}(\pi_1 a) = \bigcup_{a' \in \mathcal{A}} \{\text{Norm}(\pi_1 a')\} \\ \Gamma \vdash \text{Reductions}(\pi_2 a) = \bigcup_{a' \in \mathcal{A}} \{\text{Norm}(\pi_2 a')\} \\ \Gamma \vdash \text{Types}(\pi_1 a) = \{A\} \\ \Gamma \vdash \text{Types}(\pi_2 a) = \bigcup_{a' \in \mathcal{A}} \{\text{Norm}(B[x := \pi_1 a'])\}.\end{array}}$$

(7)

All pseudo-expressions $\tau$ whose values cannot be set by Eq. 7 are set to $\text{Types}(\tau) = \{\text{no-type}\}$, $\text{Reductions}(\tau) = \emptyset$. The expressions proper (legal expressions) of $\lambda P\rho^{\Sigma\,\text{bool}}$ are all those for which no-type $\notin \text{Types}(\tau)$ and $\exists s \in \mathcal{S} \,\text{s.t.}\, \forall \tau' \in \text{Types}(\tau), \tau' : s$ (an expression can reduce to either types/type-constructors or terms belonging to types, not both).

As a consequence of Eqs. 6 and 7, we can show that for any legal expression $\tau_1$ and reduction step $\tau_1 \rightarrow^{\rho}_{\beta} \tau_2$, we have that $\text{Types}(\tau_1) \supseteq \text{Types}(\tau_2)$, and $\text{Reductions}(\tau_1) \supseteq \text{Reductions}(\tau_2)$ (Appendix, Prop. 3). Hence, in place of a *strong preservation* property in the framework of [2] in which probabilistic reduction preserves types, we have a *weak preservation* property in which probabilistic reduction preserves a non-empty subset of types. Further, we can show that a *progress* property holds for $\rightarrow^{\rho}_{\beta}$ reduction, in that for any pseudo-expression $\tau_1$ not in $\lambda P^{\Sigma\,\text{bool}}$ normal form there exists a $\rho$ and $\tau_2$ such that $\tau_1 \rightarrow^{\rho}_{\beta} \tau_2$, and that $\sum_{\{(\rho,\tau_2)|\tau_1 \rightarrow^{\rho}_{\beta} \tau_2\}} \rho = 1$ (i.e. we can make a probabilistic reduction step, see Appendix, Prop. 4a), and that all valid reduction sequences of expressions in $\lambda P\rho^{\Sigma\,\text{bool}}$ end in an expression of $\lambda P^{\Sigma\,\text{bool}}$ in normal form (Appendix, Prop. 4b). Finally, we can show that for any $\tau_3 \in \text{Reductions}(\tau_1)$, $\tau_1 \twoheadrightarrow^{\rho}_{\beta} \tau_3$ with $\rho > 0$, and (a) for each $\tau_3 \in \text{Reductions}(\tau_1)$, $\exists \tau_4 \in \text{Types}(\tau_1) \,\text{s.t.}\, \tau_3 : \tau_4$; (b) for each $\tau_4 \in \text{Types}(\tau_1)$, $\exists \tau_3 \in \text{Reductions}(\tau_1) \,\text{s.t.}\, \tau_3 : \tau_4$, i.e. $\text{Types}(\tau_1)$ contains exactly the normal form types resulting from reductions of $\tau_1$ for which $\rho > 0$ (Appendix, Prop. 6).

Taken together, the above properties imply that *probabilistic type judgement* is possible by sampling in $\lambda P\rho^{\Sigma\,\text{bool}}$. Letting $\iota_{\tau_1}(\tau_2) = \sum_{\{\tau_3 \in \text{NF}_{\lambda P^{\Sigma\,\text{bool}}} | \tau_3 : \tau_2\}} \rho_{\tau_3}$, where $\tau_1 \twoheadrightarrow^{\rho_{\tau_3}}_{\beta} \tau_3$, we can think of $\iota_{\tau_1}(.)$ as a characteristic function for $\text{Types}(\tau_1)$, where $\iota_{\tau_1}(\tau_2) > 0 \Rightarrow \tau_2 \in \text{Types}(\tau_1)$ and $\iota_{\tau_1}(\tau_2) = 0 \Rightarrow \tau_2 \notin \text{Types}(\tau_1)$. We can estimate $\iota_{\tau_1}(\tau_2)$ by repeated $\beta\rho$ reduction of $\tau_1$ to $\tau_3$, taking the frequency with which $\tau_3 : \tau_2$ by using the fact that derivability of type judgments is decidable in $\lambda P^{\Sigma\,\text{bool}}$ (Appendix, Prop. 7).

Expressions in the system $\lambda P\rho^{\Sigma\,\text{bool}}$ can be given a two-level semantics in terms of probability distributions. Most directly, expressions can place a low level distribution across terms which they reduce to of several types, which induces a higher level distribution across the types themselves ($\iota_{\tau_1}(.)$ above). Alternatively, since expressions in $\lambda P^{\Sigma\,\text{bool}}$ may also reduce to types and type-constructors belonging to *kinds* such as $*$ or $A \rightarrow *$, they may induce a low level distribution across the former entities, and a higher level distribution across kinds. In each case, reduction of an expression corresponds to sampling explicitly from the low level distribution and implicitly from the high level distribution (a *sampling semantics*). We briefly note some special cases here (assuming an empty context). If $\text{Types}(\tau) = \{A\}$, $A : *$, $\tau$ represents a (non-dependent) distribution over a single type. When $A : \Box$, $\tau$ is a distribution over a kind, for instance a distribution over types themselves ($A = *$), or type-constructors ($A = B \rightarrow *$). If $\text{Types}(\tau) = \{A, B, C\}$, $A, B, C : *$, then $\tau$ is distribution over terms of types $A, B$ and $C$ at the low level, and a distribution over the types at the higher level. If in addition $\tau$ has the form $Fx$, where $\text{Types}(x) = \{X\}$, then $\tau$ can be thought of as a mixture distribution, where the components are determined by the distribution over $X$ denoted by $x$. Alternatively, for $\text{Types}(\tau) = \{\prod x : A.B_n, n = 1...N_1\}$, and $B_n[x := a] : *$, $\tau$ can be thought of both as a distribution across functions with types in $\text{Types}(\tau)$, and as a function itself, which when given a distribution $\tau'$ across $A$ returns a distribution across the return types of the functions in $\text{Types}(\tau)$. We note also how expressions of the form $(\lambda x.Z)\tau$ can be used to transform a general distribution across terms of several types represented by $\tau$ into another general distribution across terms of several types.

## 4 Relationship to Markov Logic Networks via Curry-Howard Correspondence

We now discuss the relationship between a particular class of PDTS, which we shall call *Dependent Type Networks* (DTNs), and Markov Logic Networks (MLNs) as introduced in [4]. As we show, these systems are equivalent under a probabilistic analogue of the formulae as types interpretation (or Curry-Howard correspondence) for typed lambda systems. The equivalence provides a further semantics for this class of PDTS in terms of MLNs, which in turn can be given factor graph semantics. Further, it is suggestive of algorithmic possibilities in that algorithms for performing inference in either of the representations can be transformed to perform inference in the other.

### 4.1 Markov Logic Networks

As defined in [4], a Markov Logic Network $L$ is a set of pairs $(F_i, w_i)$, where $F_i$ is a formula in first order logic (constructed using constants, variables, functions, predicates and logical connectives) and $w_i$ is a real number. $L$ acts as a template, which when combined with a finite set of constants $C = \{c_1, c_2, ..., c_{|C|}\}$ (which includes and may extend the constants used to build the $F_i$'s, defining the domain over which the model ranges) constructs a *ground Markov Network*, $M_{L,C}$. $M_{L,C}$ contains a binary node for every possible grounding of the predicates in $L$ (a *ground predicate* being a formula involving only the applications of a predicate to a tuple of constants). For instance, if $P_1$ is a unary predicate, $M_{L,C}$ contains a node for each application $P_1(c_i), i = 1...|C|$, and if $P_2$ is a binary predicate, a node for each application $P_2(c_i, c_j), i, j = 1...|C|$. The network also contains cliques for all possible groundings of each formula $F$ in $L$ (i.e. substitutions of constants for free variables in $F$)[5], where the clique derived from the $j$'th grounding of formula $i$ is associated with a clique potential $\psi_{i,j}$ which is 1 if the ground formula is not satisfied, and $e^{w_i}$ if it is. A *possible world* is fixed by a joint setting of the nodes of $M_{L,C}$ (under the assumptions of unique names, domain closure and known functions, see [4]), which we denote by $\mathbf{x} \in \{0,1\}^P$, where $P$ is the number of ground predicates (nodes) in the network. $M_{L,C}$ thus defines a distribution over possible worlds, which can be expressed as a random variable $X$ which ranges over the settings $\mathbf{x}$:

$$P(X = \mathbf{x}) = \frac{1}{Z}\prod_{ij}\psi_{ij}(\mathbf{x}_{ij}) = \frac{1}{Z}\prod_{i}(e^{w_i})^{n_i(\mathbf{x})}, \tag{8}$$

where $Z$ is the partition function $Z = \sum_{\mathbf{x}}\prod_{ij}\psi_{ij}(\mathbf{x}_{ij})$, and $n_i(\mathbf{x})$ denotes the number of true groundings of formula $F_i$ in joint setting $\mathbf{x}$. We note that the distribution defined by Eq. 8 is maximum entropy in the sense that any ground predicates which are independent of the weighted $F_i$'s (hence *probabilistically undecidable*) will be true or false with equal probability.

For convenience, we consider only unconditioned inference. Here, we are interested in the probability that a given formula holds in a possible world distributed according to Eq. 8, which can be written $P(F|L,C) = \sum_{\mathbf{x}\in\mathcal{X}_F} P(\mathbf{x}|L,C)$, where $\mathcal{X}_F$ is the subset of possible worlds in which query formula $F$ holds (conditional queries can be simulated by adding the conditioning formula $F'$ to the MLN with a weight tending to $\infty$). We give two examples below. First, consider an MLN $L$ with a formula $F_1$ corresponding to $A(x) \to B(x)$ (equivalently $B(x) \vee \neg A(x)$) with weight $w_1$, where $A, B$ are unary predicates and $\to$ implication, and $F_2 = A(c_1)$ with weight $w_2$ and $c_1$ a constant ($w_1, w_2 > 0$). We then consider the grounded network over over $C = \{c_1\}$, and the query $B(c_1)$. Writing $[b_1 b_2]$ for the setting of $\mathbf{x}$ which implies $A(c_1)$ when $b_1 = 1$ and $\neg A(c_1)$ when $b_1 = 0$ and similarly for $B(c_1)$ and $b_2$, the query has the probability:

$$P(B(c_1)|L,C) = P(\mathbf{x} = [0\ \ 1]) + P(\mathbf{x} = [1\ \ 1]) = \frac{e^{w_1}(1 + e^{w_2})}{e^{w_1}(2 + e^{w_2}) + e^{w_2}}, \tag{9}$$

[5] Formulas of the form $\forall x F$ are grounded as a conjunction of all possible groundings of $x$ in $F$, and $\exists x F$ as a disjunction of all possible groundings. We will not assume that $L$ is automatically transformed into conjunctive normal form, and so distinguish between $L = \{(F_1, 0.5 * w), (F_2, 0.5 * w)\}$ and $L = \{(F_1 \wedge F_2, w)\}$, as well as $L = \{(\forall x F, |C|w)\}$ and $L = \{(F, w)\}$ $(FV(F) = \{x\})$, where the latter can be taken as shorthand for $L = \{(F[x := c_1], w), (F[x := c_2], w)...(F[x := c_{|C|}], w)\}$ (which is only equivalent to $L = \{(\forall x F, |C|w)\}$ if transformation to conjunctive normal form is assumed).

which tends to 1 as $w_1, w_2$ tend to $\infty$. As a second example, consider an MLN $L$ with a formula $F_1 = A(x)$, with weight $w_1 > 0$ and $A(x)$ a unary predicate. We then consider the grounded network over $C = \{c_1, c_2\}$, and the query $\exists x A(x)$ (equivalently $A(c_1) \vee A(c_2)$, under the assumptions above). Writing $[b_1 b_2]$ for the setting of $\mathbf{x}$ which implies $A(c_1)$ when $b_1 = 1$ and $\neg A(c_1)$ when $b_1 = 0$ and similarly for $A(c_2)$ and $b_2$, the query has the probability:

$$\begin{aligned} P(\exists x A(x)|L, C) &= P(\mathbf{x} = [1\ \ 0]) + P(\mathbf{x} = [0\ \ 1]) + P(\mathbf{x} = [1\ \ 1]) \\ &= \frac{e^{w_1}(2 + e^{w_1})}{e^{w_1}(2 + e^{w_1}) + 1}, \end{aligned} \tag{10}$$

which again tends to 1 as $w_1$ tends to $\infty$. As detailed in [4], queries in larger networks can be estimated by building a factor graph representation of the ground Markov Network, and using algorithms such as Gibbs sampling, Loopy Belief Propagation and variational methods such as mean-field message passing.

### 4.2 Dependent Type Networks

We now define the notion of a *Dependent Type Network* (DTN), which will be a probabilistic dependent type system in the sense of Sec. 3. A 'language' for the PDTS will be fixed by specifying a context of the form (assuming only unary and binary predicates/functions):

$$\begin{aligned} \mathit{\Gamma}_0 &= \mathit{\Gamma}', \mathit{\Gamma}'' \\ \mathit{\Gamma}' &= < A : *, \bot : *, c_{1...N_C} : A, B^1_{1...N_{B^1}} : A \rightarrow *, \\ &\quad B^2_{1...N_{B^2}} : A \times A \rightarrow *, g^1_{1...N_{g^1}} : A \rightarrow A, g^2_{1...N_{g^2}} : A \times A \rightarrow A > \\ \mathit{\Gamma}'' &= < b^1_{n=1...N_{B^1}, m=1...N_C, 1} : B^1_n(c_m), b^1_{n=1...N_{B^1}, m=1...N_C, 0} : B^1_n(c_m) \rightarrow \bot, \\ &\quad b^2_{n=1...N_{B^2}, m=1...N_C, l=1...N_C, 1} : B^2_n(c_m, c_l), \\ &\quad b^2_{n=1...N_{B^2}, m=1...N_C, l=1...N_C, 0} : B^2_n(c_m, c_l) \rightarrow \bot > . \end{aligned} \tag{11}$$

Here, $A$ specifies a domain, to which constants $c_{1...N_c}$ belong. $B^1_{n=1...N_{B^1}}$, and $B^2_{n=1...N_{B^2}}$ will be used as unary and binary predicates over the domain respectively, $g^1_{n=1...N_{g^1}}$ and $g^2_{n=1...N_{g^2}}$ as unary and binary functions, $\bot$ to represent a contradiction, and the sets of terms $b^1$ and $b^2$ as ground predicates, while we write $B(c)$ as alternative notation for application. A DTN is fixed by augmenting this context with a further set of 'formulas' in the language, specified by $\mathit{\Gamma}_1 = \mathit{\Gamma}_0, < f_1 : F_1, ..., f_{N_F} : F_{N_F} >$. Here, a given formula such as $B^1(c_1) \wedge B^1(c_2)$ is represented by the type corresponding to the formula by the formulae as types interpretation of typed lambda systems (equivalently, the *propositions as types* interpretation, see [8]), $B^1(c_1) \times B^1(c_2)$. In general, $X \wedge Y$ corresponds to $X \times Y$ ($X$ and $Y$ being ground predicates), $X \vee Y$ to the *disjoint product* $\sum x : \mathrm{bool}.(\mathrm{if}\, x \,\mathrm{then}\, X \,\mathrm{else}\, Y)$, $X \rightarrow Y$ to $X \rightarrow Y$, $\neg X$ to $X \rightarrow \bot$, $\forall x(X(x))$ to $\prod x : A.X(x)$, and $\exists x(X(x))$ to $\sum x : A.X(x)$. Quantifications over binary and higher order predicates can be represented by nested dependent sums and products. An expression which can be typed to a formula type corresponds to a proof of that formula, and thus to asserting the formula itself (a formula is 'true' if it is inhabited). We will further equip a DTN with a set of probabilistic weights, $P = \{p_1, p_2, ..., p_{N_F}\}$, where $p_i$ will represent the probability that $F_i$ is true as a constant in the meta-language, taking values in $(0\ \ 1)$. We summarize the above in the following definition:

**Definition 1.** *A* Dependent Type Network *is a probabilistic dependent type system equipped with a context $\mathit{\Gamma}_0$ representing a language containing a domain, constants, predicates, functions, contradiction and ground predicate symbols as above, a context $\mathit{\Gamma}_1$ which augments $\mathit{\Gamma}_0$ with a set of with a set of formula proofs represented as constants assigned to types in the language ($f_i : F_i, i = 1...N_F$), and a set $P$ of meta-language constants $p_i$, which assign a probabilistic weight in $(0\ \ 1)$ to each formula.*

Given a DTN as defined, we will represent a query formula, for instance $Q = B^1_1(c_3)$, as an expression (/program) $q_Q$ where $\mathrm{Types}(q_Q) = \{B^1_1(c_3), B^1_1(c_3) \rightarrow \bot, \bot\}$. That is, the expression reduces to either a proof of $Q$, a proof of $\neg Q$, or a proof of a contradiction (the last possibility corresponding to an inconsistent world). We will canonically define $q_Q$ for a particular $Q$ (where $Q$ is a type) as follows:

$$\begin{aligned} q_Q = & \text{ let } x_1 = \text{random}_{p_1}(\lambda x : \text{bool} \,.\, \text{if } x \text{ then } f_1 \text{ else } 1) \text{ in} \\ & ... \\ & \text{let } x_{N_F} = \text{random}_{p_{N_F}}(\lambda x : \text{bool} \,.\, \text{if } x \text{ then } f_{N_F} \text{ else } 1) \text{ in} \\ & \text{let } x_{N_F+1} = D \text{ in} \\ & (\lambda x . Z_Q)(x_1, x_2, ..., x_{N_F}, x_{N_F+1}) \end{aligned} \tag{12}$$

In Eq. 12, $D$ is defined as follows:

$$\begin{aligned} D = & \text{ let } x_{1,1} = \text{random}_{0.5}(\lambda x : \text{bool} \,.\, \text{if } x \text{ then } b_{1,1,1} \text{ else } b_{1,1,0}) \text{ in} \\ & ... \\ & \text{let } x_{N_{B^1},N_C} = \text{random}_{0.5}(\lambda x : \text{bool} \,.\, \text{if } x \text{ then } b_{N_{B^1},N_C,1} \text{ else } b_{N_{B^1},N_C,0}) \text{ in} \\ & \text{let } x_{1,1,1} = \text{random}_{0.5}(\lambda x : \text{bool} \,.\, \text{if } x \text{ then } b_{1,1,1,1} \text{ else } b_{1,1,1,0}) \text{ in} \\ & ... \\ & \text{let } x_{N_{B^1},N_C,N_C} = \text{random}_{0.5}(\lambda x : \text{bool} \,.\, \text{if } x \text{ then } b_{N_{B^2},N_C,N_C,1} \text{ else } b_{N_{B^2},N_C,N_C,0}) \text{ in} \\ & (x_{1,1}, ..., x_{N_{B^1},N_C}, x_{1,1,1}, ..., x_{N_{B^2},N_C,N_C}) \end{aligned} \tag{13}$$

representing an expression which randomly samples a possible world. The term $Z_Q$ in Eq. 12 is constructed as a list of pairs $[(t, \tau_t), t \in \text{Types}(T)]$, where $T$ is:

$$\begin{aligned} T = & \text{ let } x_1 = \text{random}_{p_1}(\lambda x : \text{bool} \,.\, \text{if } x \text{ then } f_1 \text{ else } 1) \text{ in} \\ & ... \\ & \text{let } x_{N_F} = \text{random}_{p_{N_F}}(\lambda x : \text{bool} \,.\, \text{if } x \text{ then } f_{N_F} \text{ else } 1) \text{ in} \\ & \text{let } x_{N_F+1} = D \text{ in} \\ & (x_1, x_2, ..., x_{N_F}, x_{N_F+1}) \end{aligned} \tag{14}$$

For each $t \in \text{Types}(T)$, we consider the context $\Gamma_t = \Gamma', x : t$, and construct the sets $R_{Q,t}$ of expressions $r$ such that $\Gamma_t \vdash r : Q$, $S_{Q,t}$ of expressions $s$ such that $\Gamma_t \vdash s : Q \to \bot$, and $K_{Q,t}$ of expressions $k$ such that $\Gamma_t \vdash k : \bot$. If $K \neq \emptyset$, we set $\tau_t = k$ for an arbitrary $k \in K$. Otherwise, we must have either $R = \emptyset$ or $S = \emptyset$ (since if neither is the case $(sr) \in K$ for arbitrary $s \in S$, $r \in R$, and the term $D$ in Eq. 14 ensures at least one is non-empty). If $R = \emptyset \wedge K = \emptyset$, we set $\tau_t = s$ for an arbitrary $s \in S$, and if $S = \emptyset \wedge K = \emptyset$, we set $\tau_t = r$ for an arbitrary $r \in R$. $(\lambda x . Z_Q)$ thus returns a proof of $Q$, $\neg Q$, or $\bot$, depending on which of these can be constructed given the type of the input argument passed. We define the probability of a query in terms of the probability that $q_Q$ reduces by $\beta\rho$-reduction to a term $\tau$ of type $Q$ given that it does not reduce to an inconsistency (where $\tau \in \text{NF}_{\lambda P^{\Sigma}\text{bool}}$):

$$P(Q|\Gamma_0, \Gamma_1, P) = \frac{P(q_Q \twoheadrightarrow^{\rho}_{\beta} \tau : Q)}{1 - P(q_Q \twoheadrightarrow^{\rho}_{\beta} \tau : \bot)}. \tag{15}$$

The probability in Eq. 15 can be evaluated by sampling repeated reductions of Eq. 12, and rejecting those samples returning $\tau : \bot$. Writing $x_W$ for a type $x_W \in \text{Types}(D)$, i.e. a possible world, it can be shown (see Appendix, Prop. 8) that:

$$P(x_W|\Gamma_{0,1}, P) \propto \sum_{H \in \mathcal{P}(\{1...N_F\})} (\prod_{j \in H} p_j)(\prod_{j \notin H} (1 - p_j))[\forall j \in H, \exists y \text{ s. t. } \Gamma', t : x_W \vdash y : F_j], \tag{16}$$

where $[.]$ is the indicator function, which is 1 for a true condition and 0 otherwise, and $\mathcal{P}(.)$ the powerset operator. Eq. 16 implies that the probability for a world under a DTN is proportional to the combined probabilities of all subsets of initial formulae it is consistent with.

We can reformulate the examples from the end of Sec. 4.1 as DTNs. For the first, we fix a language via $\Gamma' =< A : *, \bot : *, c_1 : A, B^1_1 : A \to *, B^1_2 : A \to * >$, i.e. with one constant and two unary predicates. We then set the formulae $\Gamma_1 = \Gamma_0, < f_1 : (B^1_1(c_1) \to B^1_2(c_1)), f_2 : B^1_1(c_1) >$ with the probabilities $P = \{p_1 = (1 - e^{-w_1}), p_2 = (1 - e^{-w_2})\}$. It can be shown (following the proof of Prop. 1, part (a) below) that the distribution over worlds is identical to that in Sec. 4.1, and in particular $P(B^1_2(c_1)|\Gamma_{0,1}, P) = \frac{e^{w_1}(1+e^{w_2})}{e^{w_1}(2+e^{w_2})+e^{w_2}}$ as in Eq. 9 ($w_1, w_2 > 0$). For the second example, we let $\Gamma' =< A : *, \bot : *, c_1 : A, c_2 : A, B^1_1 : A \to * >$, and

$\mathit{\Gamma}_1 = \mathit{\Gamma}_0, < f_1 : (B^1_1(c_1)), f_2 : (B^1_1(c_2) >$, with probabilities $P = \{p_1 = p_2 = (1 - e^{-w_1})\}$. Again, we can show that the distribution over worlds is equivalent to that in Sec. 4.1. Here, we can write the query $\exists x B^1_1(x)$ using the dependent type $Q = \sum x : A.B^1_1(x)$ (i.e. we desire a pair consisting of $x$, an element of the domain, and a proof that it satisfies predicate $B^1_1$). As in Eq. 10, $P(\sum x : A.B^1_1(x)|\mathit{\Gamma}_{0,1}, P) = \frac{e^{w_1}(2+e^{w_1})}{e^{w_1}(2+e^{w_1})+1}$, $(w_1 > 0)$.

## 4.3 Relating MLNs and DTNs

We now give a proof of the equivalence of the notions of MLNs and DTNs:

**Proposition 1 (Theorem 3 in main text).** *The class of distributions over worlds representable by groundings of an MLN is the same as the class of distributions over worlds representable by DTNs. In particular we have:*
*(a) a canonical translation from ground Markov Networks $M_{L,C}$ to DTNs $\{\mathit{\Gamma}_0, \mathit{\Gamma}_1, P\}$; and*
*(b) a canonical translation from a DTNs to ground Markov Networks,*
*each preserving distributions across worlds.*

*Proof of Proposition 1, part a:* We provide here a canonical translation from an arbitrary ground MLN $M_{L,C}$ and an equivalent DTN. We assume that $L$ has been expanded so that any pair $(F, w)$ where $F$ contains free variables has been replaced by $(F_j, w), j = 1...J$ with $F_j$ ranging over all possible groundings of the free variables. We will write $N_F$ for the number of (formula,weight) pairs in $L$, $N_C$ for the number of constants in $C$, and $\mathbf{x} \in \mathbb{B}^P$ for a possible world represented as in Sec. 4.1 as a binary vector which specifies the truth or falsity of each ground predicate $p = 1...P$.

We begin by transforming $L$ to $L'$, where, for pair $(F_i, w_i) \in L$ we set $(F'_i, w'_i) \in L'$ with $F'_i = F_i$, $w' = w$ if $w \leq 0$, and $F'_i = \neg F_i$, $w' = -w$ otherwise. This transformation will preserve the distribution across worlds, since the ratio $\psi_i(\mathbf{x}_1)/\psi_i(\mathbf{x}_2)$ for a world which satisfies $F_i$, $\mathbf{x}_1$, and one which does not, $\mathbf{x}_2$, is $e^w$ in both networks $M_{L,C}$ and $M_{L',C}$.

We then construct a DTN by forming $\mathit{\Gamma}_0 = \mathit{\Gamma}', \mathit{\Gamma}''$ with the same number of constants, predicates and functions and matching arities as $M_{L,C}$, and setting $\mathit{\Gamma}_1 = \mathit{\Gamma}_0, < f_i : F''_i, i = 1 : N_F >$, for $F''_i = \hat{F}'_i \to \perp$, and $P = \{p_i = (1-\exp(w'_i)), i = 1 : N_F\}$, where we write $\hat{F}$ for the translation of formulae $F$ into a DTN type expression, using the matching constants, predicates and functions from $\mathit{\Gamma}_0$, and the formulae as types correspondence (discussed in Sec. 4.2) to translate the logical symbols into matching type constructors.

We now demonstrate that the DTN constructed above gives the same distribution across possible worlds as $M_{L,C}$. For a given world $\mathbf{x}$, write $G(\mathbf{x})$ for the proposition $G_1(\mathbf{x}) \wedge G_2(\mathbf{x}) \wedge ... G_P(\mathbf{x})$, where for ground predicate $h_p$, $G_p(\mathbf{x}) = h_p$ if $h_p$ is true in world $\mathbf{x}$, and $G_p(\mathbf{x}) = \neg h_p$ otherwise. Further, let $R_\mathbf{x} = \{i \in \{1...N_F\}|G(\mathbf{x}) \to \hat{F''_i}\}$ and $S_\mathbf{x} = \{i \in \{1...N_F\}|G(\mathbf{x}) \to \neg\hat{F''_i}\}$, where $\hat{F''_i}$ is the translation of type $F''_i$ to a logical formula by the formulae as types correspondence ($R$ is thus the set indices of formulae in $\mathit{\Gamma}_1$ which are consistent with world $\mathbf{x}$, and $S$ the set of indices of formulae that are inconsistent). Then, from Eq. 16, we have:

$$\begin{aligned}
P(\hat{G}(\mathbf{x})|\mathit{\Gamma}_{0,1}, P) \propto & \sum_{H \in \mathcal{P}(\{1...N_F\})} (\prod_{j \in H} p_j)(\prod_{j \notin H} (1 - p_j))[\forall j \in H, \exists y \, \text{s. t.} \\
& \mathit{\Gamma}', t : \hat{G}(\mathbf{x}) \vdash y : F''_j] \\
= & \sum_{r \subset R_\mathbf{x}} \prod_{i \in R_\mathbf{x}} ((p_i)^{[i \in r]}(1 - p_i)^{[i \notin r]}) \cdot \prod_{i \in S_\mathbf{x}} (1 - p_i) \\
= & \, 1 \cdot \prod_{i \in S_\mathbf{x}} (1 - p_i) \\
= & \prod_{i \in S_\mathbf{x}} \exp(w'_i). \qquad (17)
\end{aligned}$$

Similarly, for $M_{L',C}$ (and thus for $M_{L,C}$ as discussed above) we have:

$$\begin{aligned}
P(\mathbf{x}|L', C) \propto & \prod_i (e^{w'_i})^{[i \in S_\mathbf{x}]} \\
= & \prod_{i \in S_\mathbf{x}} \exp(w'_i), \qquad (18)
\end{aligned}$$

thus giving rise to identical distributions.

*Proof of Proposition 1, part b:* We suppose we have a DTN specified by $\mathit{\Gamma}_0, \mathit{\Gamma}_1, P$, where $\mathit{\Gamma}_1$ contains statements $f_i : F_i$ for $i = 1...N_F$. We use $\mathbf{x} \in \mathbb{B}^P$ as above to represent a possible world, by considering all possible groundings of the predicates in $\mathit{\Gamma}_0$. Letting $S_{\mathbf{x}} = \{i \in \{1...N_F\} | G(\mathbf{x}) \to \neg\hat{F}_i\}$ (where $\hat{F}_i$ is the translation of type $F_i$ to a logical formula by the formulae as types correspondence), and following the same derivation as Eq. 17 lines 1-3, we have:

$$P(\hat{G}(\mathbf{x})|\mathit{\Gamma}_{0,1}, P) \propto \prod_{i \in S_{\mathbf{x}}} (1 - p_i). \tag{19}$$

Consider now a grounded MLN formed using a language containing the same constants, predicates and functions as the DTN above, where we set $L = \{(\neg\hat{F}_i, \log(1 - p_i)), i = 1...N_F\}$. By Eq. 8 we will have

$$\begin{aligned} P(\mathbf{x}|L, C) &= \frac{1}{Z} \prod_i (e^{\log(1-p_i)})^{[i \in S_{\mathbf{x}}]} \\ &\propto \prod_{i \in S_{\mathbf{x}}} (1 - p_i), \end{aligned} \tag{20}$$

hence we have identical distributions.

□

Finally, we note that since MLNs provide a universal representation for finite discrete distributions, we have:

**Corollary 1.** *Both MLNs and DTNs are universal representations for distributions across finitely many discrete variables.*

*Proof.* [4] show that MLNs are such a universal representation, and so by Prop. 1 and the fact that each contain only finitely-many discrete variables (/ground predicates), the classes of representable distributions must be all finite discrete distributions for both DTNs and MLNs. In particular, the translation scheme provided in the proof of Prop. 1 part (a) applied to the canonical MLN representation of a general finite discrete distribution given in [4] provides a canonical DTN representation of the same distribution.

□

## 5 Discussion

We have used a restricted system here to show that stochastic functions can be embedded in dependent type theory (DTT), and how this naturally leads to a probabilistic logic equivalent to Markov Logic Networks. In general, we would wish to make use of the full resources of DTT [10, 11], such as inductive and recursive definitions, identity types and higher inductive types to formulate more general PDTS's, in which richer probabilistic logics may be embedded. We note however that the current framework is suggestive of a number of directions for investigation both with respect to the specific models investigated here and more generally. For instance, the relationship between DTNs and MLNs suggests that automatic theorem proving techniques may be generally applied in the context of inference in graphical models (which may naturally be expressed as MLNs), since Eq. 15 may be evaluated by a meta-algorithm which repeatedly samples assumptions and searches for proofs of $Q$, $\neg Q$ or $\perp$. The recent application of deep-learning techniques to automated and interactive theorem proving [12, 13] is potentially a powerful tool in this context.

Another suggestive direction is model learning. Recent approaches have shown that deep-learning methods can be fully integrated with probabilistic programming [14]. Our PDTS framework is naturally treated as a probabilistic programming framework, and we can straightforwardly incorporate deep neural networks as stochastic or deterministic functions within the PDTS outlined. In the logical context, this corresponds to learning formulae of arbitrary complexity in a DTN. Additionally, the ambient PDTS of our framework can naturally represent both models and inference procedures applied to models as stochastic programs, as in [14].

Finally, we note that fully integrating dependent types in probabilistic programming languages has the potential to draw a tight connection between machine learning techniques and recent computational approaches in the foundations of mathematics, such as UniMath [15].

## A Appendix

Proofs of results mentioned in Sec. 2 and 3 are given below.

**Proposition 2.** *$\beta$-reduction in $\lambda P^{\Sigma\,\mathrm{bool}}$ preserves types up to $\beta$ equivalence.*

*Proof.* We first show the proposition folds for all rules in Eq. 3. From the products(1) and products(2) rules in Eq. 4 we have:

$$\begin{aligned}\pi_1(U_1, U_2)_{\sum x:A.B} : A \rightarrow_\beta U_1 : A\\ \pi_2(U_1, U_2)_{\sum x:A.B} : B[x := U_1] \rightarrow_\beta U_2 : B[x := \pi_1(U_1, U_2)_{\sum x:A.B}],\end{aligned} \tag{21}$$

where, we note that $B[x := U_1] =_\beta B[x := \pi_1(U_1, U_2)_{\sum x:A.B}]$. Further, if we have $\tau_1 : B[x = \mathrm{true}]$ and $\tau_2 : B[x = \mathrm{false}]$, from the if rule in Eq. 4:

$$\begin{aligned}(\mathrm{if\,true\,then}\,\tau_1\,\mathrm{else}\,\tau_2) : B[x = \mathrm{true}] \rightarrow_\beta \tau_1 : B[x = \mathrm{true}]\\ (\mathrm{if\,false\,then}\,\tau_1\,\mathrm{else}\,\tau_2) : B[x = \mathrm{false}] \rightarrow_\beta \tau_2 : B[x = \mathrm{false}],\end{aligned} \tag{22}$$

and from the abstraction and application rules in Eq. 4, letting $(\lambda x : \tau_1.\tau_2) : \prod x : A.B$:

$$((\lambda x : \tau_1.\tau_2)\tau_3) : B[x := \tau_3] \rightarrow_\beta \tau_2[x := \tau_3] : B[x := \tau_3]. \tag{23}$$

The above cover the base cases of the $\beta$-reduction relation. For $\tau_1 \rightarrow_\beta \tau_1'$ where $\tau_2 \rightarrow_\beta \tau_2'$, and $\tau_1'$ is the result of replacing free sub-expression $\tau_2$ for $\tau_2'$ in $\tau_1$, we prove the result by induction on the structure of $\tau_1$. First we outline a general induction step for each form that $\tau_1$ may take. For $\tau_1 = \prod_x : A.B : s$, with either the inductive hypothesis $A : * \rightarrow_\beta A' : *$ or $B : s \rightarrow_\beta B' : s$, we have that $\tau_1' = \prod x : A'.B : s$ or $\tau_1' = \prod x : A.B' : s$ respectively by the type/kind formation rule in Eq. 4 (similarly for $\Sigma$-types). For $\tau_1 = (Fa) : B[x := a]$ with inductive hypothesis $F : (\prod x : A.B) \rightarrow_\beta F' : (\prod x : A.B)$ or $a : A \rightarrow_\beta a' : A$ (where we implicitly use the type conversion rule, Eq. 4, to equalize types which are $\beta$-equivalent), we have respectively $\tau_1' = F'a : B[x := a]$ and $\tau_1' = Fa' : B[x := a'] =_\beta B[x := a]$ by the application rule in Eq. 4. For $\tau_1 = (\lambda x : A.b) : (\prod x : A.B)$ and inductive hypothesis $A : s \rightarrow_\beta A' : s$ or $b : B \rightarrow_\beta b' : B$, we have respectively $\tau_1' = (\lambda x : A'.b) : (\prod x : A'.B) =_\beta (\prod x : A.B)$ and $\tau_1' = (\lambda x : A.b') : (\prod x : A.B)$ by the abstraction rule in Eq. 4. For $\tau_1 = \mathrm{if}\,a\,\mathrm{then}\,b_1\,\mathrm{else}\,b_2$ and inductive hypothesis $b_1 : B[x := \mathrm{true}] \rightarrow_\beta b_1' : B[x := \mathrm{true}]$ or $b_2 : B[x := \mathrm{false}] \rightarrow_\beta b_2' : B[x := \mathrm{false}]$, we have respectively $\tau_1' = \mathrm{if}\,a\,\mathrm{then}\,b_1'\,\mathrm{else}\,b_2 : B[x := a]$ and $\tau_1' = \mathrm{if}\,a\,\mathrm{then}\,b_1\,\mathrm{else}\,b_2' : B[x := a]$ by the if rule in Eq. 4. Other typing rules in Eq. 4 do not involve sub-expressions, so need not be considered (except the product constant type-tags in the products (1) rule, which can be handled via the type conversion rule).

For the inductive argument, we begin by considering reductions from expressions $\tau_1$ whose maximum sub-expression nesting depth is 1. Here, $\tau_2$ will correspond to an expression nested directly below $\tau_1$, and thus the inductive hypotheses in all cases considered will be satisfied by one of the base cases in Eqs. 21, 22 and 23. For $\tau_1$ with maximum sub-expression nesting depth $n$ greater than 1, $\tau_2$ will be nested inside a set of sub-expressions of $\tau_1$. Since these will all have nesting depths less than $n$, the inductive hypothesis can be applied to them if we replace $\tau_2$ by $\tau_2'$, and hence also for $\tau_1$ by one of the cases above.

□

**Proposition 3.** *(Weak-preservation) For expressions $\tau_1$ and $\tau_1'$ in $\lambda P\rho^{\Sigma\,\mathrm{bool}}$ such that $\tau_1 \rightarrow_\beta^\rho \tau_1'$ for non-zero $\rho$, we have that* $\mathrm{Types}(\tau_1) \supseteq \mathrm{Types}(\tau_1')$*, and* $\mathrm{Reductions}(\tau_1) \supseteq \mathrm{Reductions}(\tau_1')$*, where* $\mathrm{Types}(\tau_1')$ *and* $\mathrm{Reductions}(\tau_1')$ *are non-empty.*

*Proof.* We note initially that for any expression $\tau_1$ which is identical to an expression in $\lambda P^{\Sigma\,\mathrm{bool}}$ such that $\tau_1 : \tau_2$, by Prop. 2 we have that for $\tau_1 \rightarrow_\beta^\rho \tau_1'$, $\mathrm{Reductions}(\tau_1) = \mathrm{Reductions}(\tau_1') = \{\mathrm{Norm}(\tau_1)\}$ and $\mathrm{Types}(\tau_1) = \mathrm{Types}(\tau_1') = \{\mathrm{Norm}(\tau_2)\}$. For a general expression $\tau_1 \rightarrow_\beta^\rho \tau_1'$, we check as base cases each of the reduction rules in Eq. 6. The projection rules involve only values, and so are covered by the above. For if reduction, by the if rule in Eq. 7 we have that for $\mathrm{Reductions}(\tau_1) = \{b_{1,n}, n = 1...N_{1,1}\}$,$\mathrm{Types}(\tau_1) = \{B_{1,n}, n = 1...N_{1,2}\}$,$\mathrm{Reductions}(\tau_2) = \{b_{2,n}, n = 1...N_{2,1}\}$,$\mathrm{Types}(\tau_2) = \{B_{2,n}, n = 1...N_{2,2}\}$:

$$\begin{aligned}\mathrm{Reductions}(\mathrm{if\,true\,then}\,\tau_1\,\mathrm{else}\,\tau_2) = \mathrm{Reductions}(\tau_1) = \{b_{1,n}, n = 1...N_{1,1}\}\\ \mathrm{Reductions}(\mathrm{if\,false\,then}\,\tau_1\,\mathrm{else}\,\tau_2) = \mathrm{Reductions}(\tau_2) = \{b_{2,n}, n = 1...N_{2,1}\}\\ \mathrm{Types}(\mathrm{if\,true\,then}\,\tau_1\,\mathrm{else}\,\tau_2) = \mathrm{Types}(\tau_1) = \{B_{1,n}, n = 1...N_{1,2}\}\\ \mathrm{Types}(\mathrm{if\,false\,then}\,\tau_1\,\mathrm{else}\,\tau_2) = \mathrm{Types}(\tau_2) = \{B_{2,n}, n = 1...N_{2,2}\}.\end{aligned} \tag{24}$$

For random expressions, by the random and application (1) rules in Eq. 7 we have for $\text{Reductions}(F) = \{\lambda x : \text{bool}\,.a_n, n = 1...N_1\}$, $\text{Types}(F) = \{\prod x : \text{bool}\,.A_n, n = 1...N_2\}$:

$$\begin{aligned}
\text{Reductions}(F\,\text{true}) &= \{\text{Norm}(a_n[x := \text{true}]), n = 1...N_1\} \\
&\subseteq \bigcup_{b\in\{\text{true},\text{false}\}} \{\text{Norm}(a_n[x := b]), n = 1...N_1\} \\
&= \text{Reductions}(\text{random}(F)) \\
\text{Reductions}(F\,\text{false}) &= \{\text{Norm}(a_n[x := \text{false}]), n = 1...N_1\} \\
&\subseteq \bigcup_{b\in\{\text{true},\text{false}\}} \{\text{Norm}(a_n[x := b]), n = 1...N_1\} \\
&= \text{Reductions}(\text{random}(F)) \\
\text{Types}(F\,\text{true}) &= \{\text{Norm}(A_n[x := \text{true}]), n = 1...N_2\} \\
&\subseteq \bigcup_{b\in\{\text{true},\text{false}\}} \{\text{Norm}(A_n[x := b]), n = 1...N_2\} \\
&= \text{Types}(\text{random}(F)) \\
\text{Types}(F\,\text{false}) &= \{\text{Norm}(A_n[x := \text{false}]), n = 1...N_2\} \\
&\subseteq \bigcup_{b\in\{\text{true},\text{false}\}} \{\text{Norm}(A_n[x := b]), n = 1...N_2\} \\
&= \text{Types}(\text{random}(F)). \qquad (25)
\end{aligned}$$

For $\lambda$ applications of the form $(Fa)$ with $F = \lambda x : A.b$, $(b \in \mathcal{T}_x)$ and $\Gamma, x : A \vdash \text{Reductions}(b) = \{b_n, n = 1...N_1\}, \text{Types}(b) = \{B_n, n = 1...N_2\}$, where $a : A$, $\text{Reductions}(a) = \{a\}$, $\text{Types}(a) = \{A\}$, $\text{Reductions}(F) = \{\lambda a : A.b_n, n = 1...N_1\}$, $\text{Types}(F) = \{\prod a : A.B_n, n = 1...N_2\}$, we have by the application (1) and abstraction rules in Eq. 7:

$$\begin{aligned}
\text{Reductions}(Fa) &= \{\text{Norm}(b_n[x := a]), n = 1...N_1\} \\
&= \text{Reductions}(b[x := a]) \\
\text{Types}(Fa) &= \{\text{Norm}(B_n[x := a]), n = 1...N_2\} \\
&= \text{Types}(b[x := a]). \qquad (26)
\end{aligned}$$

For $\lambda$ applications of the form $(\lambda.Z)a$, where $a : A$, $\text{Reductions}(a) = \{a\}$, $\text{Types}(a) = \{A\}$, and $(A, b) \in\in Z$, $\forall b' \neq b, \neg((A, b') \in\in Z)$, where $\Gamma, x : A \vdash \text{Reductions}(b) = \{b_n, n = 1...N_1\}, \text{Types}(b) = \{B_n, n = 1...N_2\}$, $b \in \mathcal{T}_x$, we have by the application (1) and (2) rules in Eq. 7:

$$\begin{aligned}
\text{Reductions}((\lambda.Z)a) &= \{\text{Norm}(b_n[x := a]), n = 1...N_1\} \\
&= \text{Reductions}((\lambda x : A.b)a) \\
\text{Types}((\lambda.Z)a) &= \{\text{Norm}(B_n[x := a]), n = 1...N_2\} \\
&= \text{Types}((\lambda x : A.b)a). \qquad (27)
\end{aligned}$$

The above form the base cases for the reduction relation $\rightarrow^{\rho}_{\beta}$, which show that the proposition holds for each rule in Eq. 6 in the case that the sub-expression being reduced is the outermost sub-expression. For the general case (for a proper sub-expression) we can use a similar inductive argument to Prop. 1 above on the structure on $\tau$, the expression to be reduced, where the induction is on the number of nesting levels $n$ between $\tau_2$, the sub-expression to be reduced, and the outermost level (Eq. 7 providing all cases for relating sub-expressions at level $n$ to those at level $n - 1$).

Further, the $\text{Reductions}(\tau_1')$ and $\text{Types}(\tau_1')$ sets after reduction of a legal expression $\tau_1 \rightarrow^{\rho}_{\beta} \tau_1'$ must be non-empty, since $\text{Reductions}(\tau_1') = \emptyset \Rightarrow \text{Types}(\tau_1') = \{\text{no-type}\}$, and we have $\text{no-type} \notin \text{Types}(\tau_1)$ for legal $\tau_1$.

□

**Proposition 4.** *For any legal expression $\tau_1$ in $\lambda P\rho^{\Sigma\,\text{bool}}$ we have that: (a) (Progress) Either $\tau_1$ is a $\lambda P^{\Sigma\,\text{bool}}$ expression in normal form, or there exists a $\rho$ and $\tau_2$ such that $\tau_1 \rightarrow^{\rho}_{\beta} \tau_2$, and that $\sum_{\{(\rho,\tau_2)|\tau_1\rightarrow^{\rho}_{\beta}\tau_2\}} \rho = 1$; and (b) all valid reduction sequences from $\tau_1$ can be extended to a sequence ending in a $\lambda P^{\Sigma\,\text{bool}}$ normal form.*

*Proof.* For part (a) we start by observing that the proposition is already satisfied for terms which are shared between $\lambda P\rho^{\Sigma\,\text{bool}}$ and $\lambda P^{\Sigma\,\text{bool}}$ ($\tau_1 \in \mathcal{T}_{\lambda P^{\Sigma\,\text{bool}}}$) since in $\lambda P^{\Sigma\,\text{bool}}$ we have that leftmost-outermost reduction always finds a normal form by strong normalization (see [9], Theorem 5.36). The proposition automatically follows since there is a unique leftmost-outermost reduction for such terms not in normal form with $\rho = 1$ (the rules in Eq. 3 become deterministic rules in Eq. 6).

For $\tau_1$ is not in $\mathcal{T}_{\lambda P^{\Sigma\,\text{bool}}}$, we consider first the case that it contains no $\text{random}_\rho(\tau_3)$ sub-expressions. Then it must contain at least one $(\lambda x.Z)\tau_3$ sub-expression. For such a sub-expression with the lowest nesting depth, $\tau_3$ will contain no further such sub-expressions, and hence $\tau_3 \in \mathcal{T}_{\lambda P^{\Sigma\,\text{bool}}}$, and thus either this expression, or one with a higher nesting level $(\lambda x.Z)\tau_3'$ such that $\tau_3' \in \mathcal{T}_{\lambda P^{\Sigma\,\text{bool}}}$, will be the leftmost-outermost reducible expression of this form, and can be deterministically reduced according to rule 8, Eq. 6 (the full conditions of the rule following from the application (2) rule in Eq. 7 for a legal expression). Note that sub-expressions of this form are prioritized by condition (i) following Eq. 6 above free sub-expressions of other forms. Consider now the case that $\tau_1$ contains at least one $\text{random}_{\rho'}(\tau_3)$ sub-expression. Then, we will either have a leftmost-outermost reducible sub-expression of the form $(\lambda x.Z)\tau_3$, or the first $\text{random}(\tau_3)$ sub-expression encountered by leftmost-outermost order will be available for reduction by rule 5 or 6, Eq. 6, since we placed no further conditions on the reduction of such expressions and they are prioritized over free sub-expressions of other forms (following Eq. 6). Here, for the two possible reductions by rules 5 and 6, we have $\rho$ values of $\rho'$ and $1-\rho'$, and hence $\sum_{\{(\rho,\tau_2)|\tau_1 \to^{\rho}_{\beta} \tau_2\}} \rho = 1$ as required.

For part (b), we note again that this follows directly when $\tau_1 \in \mathcal{T}_{\lambda P^{\Sigma\,\text{bool}}}$ by strong normalization of $\lambda P^{\Sigma\,\text{bool}}$. For general terms of $\lambda P\rho^{\Sigma\,\text{bool}}$, part (a) of the proposition ensures that a probabilistic reduction is always possible. There can be only a finite number of sub-expressions of the form $\text{random}_{\rho'}(\tau_3)$ and $(\lambda x.Z)\tau_5$ in $\tau_1$ and singly or multiply nested within $\mathcal{Z}$ constructions of $(\lambda x.Z)\tau_5$ expressions within $\tau_1$. Combining this with the facts that, (1) from condition (i) following Eq. 6 (prioritizing the reduction of these expressions); (2) $\text{random}_{\rho'}(\tau_3)$ expressions are always reducible; (3) $(\lambda x.Z)\tau_3$ expressions are always reducible if no $\text{random}_{\rho'}(\tau_3')$ or $(\lambda x.Z)\tau_3'$ expressions are nested beneath them (see part (a) above); and (4) rules 5, 6 and 8 in Eq. 6 always decrease the total number of such expressions across all sub-expressions and $\mathcal{Z}$ constructions of $\tau_1$; we can deduce that any valid reduction sequence can be extended to (or already includes as an initial sub-sequence) one which initially reduces all such terms and ends in an expression of $\lambda P^{\Sigma\,\text{bool}}$. Such a sequence can then be extended to reach a normal form of $\lambda P^{\Sigma\,\text{bool}}$, again by strong normalization.

□

**Proposition 5.** *For legal expressions $\tau_1, \tau_2$ in $\lambda P\rho^{\Sigma\,\text{bool}}$ such that $\tau_1 \twoheadrightarrow^{\rho_0}_{\beta} \tau_2$, $\rho_0 \leq 1$.*

*Proof.* By Prop. 4, all legal expressions $\tau$ of $\lambda P\rho^{\Sigma\,\text{bool}}$ have at most two valid reductions, and all maximal sequences must terminate in a $\lambda P^{\Sigma\,\text{bool}}$ normal form. We can thus form a reduction tree for $\tau_1$, with nodes labeled by expressions ($\tau_1$ at the root, the children of a node labeled by the possible reductions of that node, and $\lambda P^{\Sigma\,\text{bool}}$ normal forms at the leaves, repeating expression labels if they are encountered in multiple reductions), and edges labeled by reduction probabilities. Since by Prop. 4(a) the sum of the edges between a single parent and its children nodes are non-negative and sum to one, writing $\rho_l$ for the product of the weights along the branch from the root leading to leaf $l$, we have that $\sum_l \rho_l = 1$. Since $\lambda P\rho^{\Sigma\,\text{bool}}$ does not have general recursion, the given expression $\tau_2$ in the proposition can occur at most once along any branch from root to leaf. Further, we have that for a given node $N$, the product of the weights on the path from the root to $N$ is $\sum_{l\in L_N} \rho_l$, where $L_N$ contains all leaves whose branches from the root pass through $N$. Hence, we have $\rho_0 = \sum_{\{l|\exists N(l\in L_N \wedge \psi(N)=\tau_2)\}} \rho_l \leq 0$, where $\psi(N)$ returns the expression with which node $N$ is labeled.

□

**Proposition 6.** *For $\tau_1$ in $\lambda P\rho^{\Sigma\,\text{bool}}$, and any $\tau_3 \in \text{Reductions}(\tau_1)$, we have that $\tau_1 \twoheadrightarrow^{\rho}_{\beta} \tau_3$ with $\rho > 0$, and (a) for each $\tau_3 \in \text{Reductions}(\tau_1)$, $\exists \tau_4 \in \text{Types}(\tau_1)\,\text{s.t.}\,\tau_3 : \tau_4$; (b) for each $\tau_4 \in \text{Types}(\tau_1)$, $\exists \tau_3 \in \text{Reductions}(\tau_1)\,\text{s.t.}\,\tau_3 : \tau_4$.*

*Proof.* We begin by noting that for expressions $\tau_1 \in \mathcal{T}_{\lambda P^{\Sigma\,\text{bool}}}$, the proposition holds, since $\text{Reductions}(\tau_1) = \{\tau_3\}$ and $\text{Types}(\tau_1) = \{\tau_4\}$ are singletons, and we can set $T_{\tau_1}(\tau_3) = \tau_4$ and $R_{\tau_1}(\tau_4) = \tau_3$, where we use $T_{\tau_1}(.)$ to denote a function which, for any $\tau_3' \in \text{Reductions}(\tau_1)$, picks out a $\tau_4' \in \text{Types}(\tau_1)$ for which $\tau_3' : \tau_4'$, and $R_{\tau_1}(.)$ to denote a function which, for any $\tau_4' \in \text{Types}(\tau_1)$, picks out a $\tau_3' \in \text{Reductions}(\tau_1)$ for which $\tau_3' : \tau_4'$.

For the general case, we use an induction on the formation rules for legal expressions in $\lambda P\rho^{\Sigma\,\text{bool}}$, Eq. 7, where in each case we assume the proposition holds for the expressions in the antecedent of the rule, and derive from this that it holds for the expression(s) in the consequent.

**(random)**: For $\tau_1 = \text{random}_\rho(F)$ we associate with $\tau_3 = \text{Norm}(a_n[x := \text{true}])$ the reduction which first reduces $\tau_1$ to $F\,\text{true}$, reduces $F\,\text{true}$ to $(\lambda x : \text{bool}\,.a_n)\,\text{true}$ (by the inductive hypothesis (IH), noting that leftmost-outermost reduction will reduce expressions in $F$ first), performs the substitution $a_n[x := \text{true}]$, and finally reduces $a_n[x := \text{true}]$ to normal form. Letting $F \twoheadrightarrow_\beta^{\rho'} \lambda x : \text{bool}\,.a_n$, we have $\tau_1 \twoheadrightarrow_\beta^{\rho\cdot\rho'} \tau_3$. Similarly, for $\tau_3 = \text{Norm}(a_n[x := \text{false}])$ we have $\tau_1 \twoheadrightarrow_\beta^{(1-\rho)\cdot\rho'} \tau_3$. Finally, for $b \in \{\text{true}, \text{false}\}$ we set $T_{\tau_1}(\text{Norm}(a_n[x := b])) = \text{Norm}(A_m[x := b])$ where $T_F(\lambda x : \text{bool}\,.a_n) = \prod x : \text{bool}\,.A_m$, and $R_{\tau_1}(\text{Norm}(A_n[x := b])) = \text{Norm}((R_F(\prod x : \text{bool}\,.A_n))b)$ (for convenience, embedding the meta-language function $R_F(.)$ in the syntax of $\lambda P\rho^{\Sigma\,\text{bool}}$).

**(if)**: For $\tau_1 = \text{if}\,a\,\text{then}\,a_1\,\text{else}\,a_2$ we associate with $\tau_3 = \text{if}\,a\,\text{then}\,b_{1,n_1}\,\text{else}\,b_{2,n_2}$ the reduction which reduces $a_1$ to $b_{1,n_1}$, and $a_2$ to $b_{2,n_2}$ (the exact sequence of reduction of sub-expressions will depend on the ordering rules following Eq. 6). Letting $a_1 \twoheadrightarrow_\beta^{\rho_1} b_{1,n_1}$, and $a_2 \twoheadrightarrow_\beta^{\rho_2} b_{2,n_2}$, we have $\tau_1 \twoheadrightarrow_\beta^{\rho_1\cdot\rho_2} \tau_3$. Finally, we set $T_{\tau_1}(\text{if}\,a\,\text{then}\,b_{1,n_1}\,\text{else}\,b_{2,n_2}) = \text{if}\,a\,\text{then}\,T_{a_1}(b_{1,n_1})\,\text{else}\,T_{a_2}(b_{2,n_2})$ and $R_{\tau_1}(\text{if}\,a\,\text{then}\,B_{1,n_1}\,\text{else}\,B_{2,n_2}) = \text{if}\,a\,\text{then}\,R_{a_1}(B_{1,n_1})\,\text{else}\,R_{a_2}(B_{2,n_2})$.

**(abstraction)**: For $\tau_1 = \lambda x : A.b$ we associate with $\tau_3 = \lambda x : A.b_n$ the reduction which prefixes all expressions in a reduction sequence from $b$ to $b_n$ (IH) with $\lambda x : A$, noting that we have that $b \in \mathcal{T}_x$, and so $b$ contains no reducible sub-expressions containing $x$. Letting $b \twoheadrightarrow_\beta^{\rho'} b_n$, we have $\tau_1 \twoheadrightarrow_\beta^{\rho'} \tau_3$. Finally, we set $T_{\tau_1}(\lambda x : A.b_n) = \prod x : A.T_b(b_n)$ and $R_{\tau_1}(\prod x : A.B_n) = \lambda x : A.R_b(B_n)$.

**(application (1))**: For $\tau_1 = Fa$ we associate with $\tau_3 = \text{Norm}(b_n[x := a'])$ the reduction which first reduces $F$ to $\lambda x : A.b_n$ with weight $\rho_1$ (IH), then $a$ to $a'$ with weight $\rho_2$ (IH), makes the substitution $b_n[x := a']$, and then reduces to normal form. For this reduction we have $\tau_1 \twoheadrightarrow_\beta^{\rho_1\cdot\rho_2} \tau_3$. Finally, we set $T_{\tau_1}(\text{Norm}(b_n[x := a'])) = \text{Norm}(B_m[x := a'])$ where $T_F(\lambda x : A.b_n) = \prod x : A.B_m$, and $R_{\tau_1}(\text{Norm}(B_n[x := a'])) = \text{Norm}((R_F(\prod x : A.B_n))a')$.

**(application (2))**: For $\tau_1 = (\lambda x.Z)a$ we associate with $\tau_3 \in \text{Reductions}(b[x := a'])$ the reduction which first reduces $a$ to $a'$ with weight $\rho_1$ (IH), then reduces $(\lambda x.Z)a'$ to $(\lambda x : A.b)a'$ where $a' : A$ and $(A, b) \in\in Z$, and finally reduces $(\lambda x : A.b)a'$ to $b[x := a']$ and then to $\tau_3$ with weight $\rho_2$ (IH). For this reduction we have $\tau_1 \twoheadrightarrow_\beta^{\rho_1\cdot\rho_2} \tau_3$. Further, we set $T_{\tau_1}(\tau_3 \in \text{Reductions}(b[x := a'])) = T_{b[x:=a']}(\tau_3) \in \text{Types}(b[x := a'])$, and $R_{\tau_1}(\tau_4 \in \text{Types}(b[x := a'])) = R_{b[x:=a']}(\tau_4) \in \text{Reductions}(b[x := a'])$.

**($\prod$ / $\sum$ formation)**: For $\tau_1 = \prod x : A.B$ we associate with $\tau_3 = \prod x : A.B_n$ the reduction which prefixes all expressions in a reduction sequence from $B$ to $B_n$ (IH) with $\prod x : A$, noting that we have that $B \in \mathcal{T}_x$, and so $B$ contains no reducible sub-expressions containing $x$. Letting $B \twoheadrightarrow_\beta^{\rho'} B_n$, we have $\tau_1 \twoheadrightarrow_\beta^{\rho'} \tau_3$. Similarly, by prefixing the same reduction sequence with $\sum x : A$, we have $\tau_1' = \sum x : A.B \twoheadrightarrow_\beta^{\rho'} \tau_3' = \sum x : A.B_n$. We can further set $T_{\tau_1}(\prod x : A.B_n) = s$, $R_{\tau_1'}(\sum x : A.B_n) = s$, $R_{\tau_1}(s) = \prod x : A.B_1$, and $R_{\tau_1'}(s) = \sum x : A.B_1$.

**(products)**: For $\tau_1 = \pi_n a$ $(n = 1, 2)$ we associate with $\tau_3 = \text{Norm}(\pi_n a')$ the reduction which first reduces $a$ to $a'$ with weight $\rho'$ (IH), and then reduces $\pi_n a'$ to normal form. For this reduction we have $\tau_1 \twoheadrightarrow_\beta^{\rho'} \tau_3$. Further, we set $T_{\pi_1 a}(\text{Norm}(\pi_1 a')) = A$, $R_{\pi_1 a}(A)$ to an arbitrary $\text{Norm}(\pi_1 a')$, $T_{\pi_2 a}(\text{Norm}(\pi_2 a')) = \text{Norm}(B[x := \pi_1 a'])$, and $R_{\pi_2 a}(\text{Norm}(B[x := \pi_1 a'])) = \text{Norm}(\pi_2 a')$.

□

**Proposition 7 (Theorem 1 in main text).** *(Probabilistic Type Judgment): Letting* $\iota_{\tau_1}(\tau_2) = \sum_{\{\tau_3 \in \text{NF}_{\lambda P^{\Sigma\,\text{bool}}} | \tau_3 : \tau_2\}} \rho_{\tau_3}$*, where* $\tau_1 \twoheadrightarrow_\beta^{\rho_{\tau_3}} \tau_3$*, we have (a)* $\iota_{\tau_1}(\tau_2) > 0 \Rightarrow \tau_2 \in \text{Types}(\tau_1)$ *and* $\iota_{\tau_1}(\tau_2) = 0 \Rightarrow \tau_2 \notin \text{Types}(\tau_1)$*; and (b)* $\iota_{\tau_1}(\tau_2)$ *can be estimated by the frequency with which* $\tau_1$ *reduces to* $\tau_3 \in \text{NF}_{\lambda P^{\Sigma\,\text{bool}}}$ *such that* $\tau_3 : \tau_2$.

*Proof.* Part (a) follows firstly from that fact that we have shown by Props. 3 and 4b that all maximal reduction sequences of $\tau_1$ end in a single member of $\text{Reductions}(\tau_1)$ having $\lambda P\rho^{\Sigma\,\text{bool}}$ normal form, and by Prop. 6 that every member of $\text{Reductions}(\tau_1)$ has a reduction from $\tau_1$ having non-zero probability, implying together that $\text{Reductions}(\tau_1)$ contains exactly the maximal reductions of all reduction sequences from $\tau_1$; and secondly from the fact that, from Prop. 6 we have that $\tau_2 \in \text{Types}(\tau_1) \Rightarrow \exists \tau_3 \in \text{Reductions}(\tau_1)\,\text{s.t.}\,\tau_3 : \tau_2$ and $\tau_3 \in \text{Reductions}(\tau_1) \Rightarrow \exists \tau_2 \in \text{Types}(\tau_1)\,\text{s.t.}\,\tau_3 : \tau_2$. Part (b) follows from the fact that all members of $\text{Reductions}(\tau_1)$ and $\text{Types}(\tau_1)$ are expressions in $\lambda P^{\Sigma\,\text{bool}}$, and that $\beta\rho$ reduction corresponds to sampling from $\text{Reductions}(\tau_1)$ according to the probabilities $\rho_{\tau_3}$ (which follows from Props. 3a, 5 and 6). Since the derivability of type judgments (the truth of statements of the form $\tau_3 : \tau_2$) is decidable in $\lambda P^{\Sigma\,\text{bool}}$ (see [9], Theorem

5.21), we can determine for a given reduction if $\tau_3 : \tau_2$, and use the frequency with which this is true to estimate $\iota_{\tau_1}(\tau_2) = \sum_{\{\tau_3 \in \mathrm{NF}_{\lambda P \Sigma\, \mathrm{bool}} | \tau_3 : \tau_2\}} \rho_{\tau_3}$.

$\square$

**Proposition 8 (Theorem 2 in main text).** *For a Dependent Type Network defined by $\Gamma_0, \Gamma_1$ and $P$ as in Definition 1 (Sec. 4.2) and $x_W \in \mathrm{Types}(D)$ representing a possible world, with D as in Eq. 13, writing $[.]$ for the indicator function which is 1 for a true condition and 0 otherwise, and $\mathcal{P}(.)$ for the powerset operator, we have:*

$$P(x_W|\Gamma_{0,1}, P) \propto \sum_{H \in \mathcal{P}(\{1 \dots N_F\})} (\prod_{j \in H} p_j)(\prod_{j \notin H} (1 - p_j))[\forall j \in H, \exists y \,\mathrm{s.t.}\, \Gamma', t : x_W \vdash y : F_j]. \tag{28}$$

*Proof.* By Eq. 15 we have that $P(x_W|\Gamma_{0,1}, P) \propto P(q_Q \twoheadrightarrow^{\rho}_{\beta} \tau : Q)$, $Q = x_w$ (where the normalizing constant, $1 - P(q_Q \twoheadrightarrow^{\rho}_{\beta} \tau : \bot) = \sum_W P(x_W|\Gamma_{0,1}, P)$, by mutual exclusion of possible worlds, $x_W$). For $q_Q$ to reduce to $\tau : Q$, two events must occur. First, $D$ in Eq. 12 (defined in Eq. 13) must reduce to a term $\tau_1 : Q$, since otherwise $\tau_1$ will contain a proof for some ground predicate which is false in $x_W$, allowing a proof of $x_W \to \bot$, and thus $q_Q$ will reduce to either $\tau : Q \to \bot$ or $\tau : \bot$. Writing $N_D$ for the size of the tuple $\tau_1 : Q$, by Eq. 13 the probability of this occurring is $P_1 = 2^{-N_D}$. Second, the formulae $F_{i=1 \dots N_F}$ for which the binding of $x_i$ in Eq. 12 reduces to $f_i : F_i$ must all be consistent with the world represented by $x_W$. Since $x_W$ fixes all ground predicates, it also fixes all formulae built from these, hence a formula is consistent with $x_W$ if $\hat{x}_W \to \hat{F}_i$ (where $\hat{x}_W$ is the formulae as types interpretation translation of $x_w$, as discussed in Sec. 4.2), or equivalently, $\exists y \,\mathrm{s.t.}\, \Gamma', t : x_W \vdash y : F_i$ ($x_W$ being inhabited implies that $F_i$ is inhabited). If $x_i$ binds to $f_i : F_i$ for an inconsistent formula, we will be able to construct a proof of $x_W \to \bot$, and hence again $q_Q$ will reduce to either $\tau : Q \to \bot$ or $\tau : \bot$. The probability of this event is $P_2 = \sum_{H \in \mathcal{P}(\{1 \dots N_F\})} P(H)[\forall j \in H, \exists y \,\mathrm{s.t.}\, \Gamma', t : x_W \vdash y : F_j]$, where $P(H)$ is the probability that for the indices $i$ in the subset $H$, $x_i$ is bound to $f_i : F_i$, and for all other formulae $x_i$ is bound to $1 : \mathrm{unit}$ (hence we sum across all possible subsets of formulae $H$, selecting only those subsets consistent with the query). Since $P(H) = (\prod_{j \in H} p_j)(\prod_{j \notin H}(1 - p_j))$, we have:

$$\begin{aligned} P(x_W|\Gamma_{0,1}, P) &\propto P_1 \cdot P_2 \\ &= 2^{-N_D} \cdot P_2 \\ &\propto P_2 \\ &= \sum_{H \in \mathcal{P}(\{1 \dots N_F\})} P(H)[\forall j \in H, \exists y \,\mathrm{s.t.}\, \Gamma', t : x_W \vdash y : F_j] \\ &= \sum_{H \in \mathcal{P}(\{1 \dots N_F\})} (\prod_{j \in H} p_j)(\prod_{j \notin H} (1 - p_j))[\forall j \in H, \exists y \,\mathrm{s.t.}\, \Gamma', t : x_W \vdash y : F_j], \end{aligned} \tag{29}$$

where the step from lines 2 to 3 follows from the fact that $N_D$ is constant for all worlds, and hence $P_1$ may be absorbed into the normalizing constant.

$\square$

## References

1. N.D. Goodman, V.K. Mansinghka, D. Roy, K. Bonawitz, and J.B. Tenenbaum. Church: A language for generative models. *UAI*, 2008.
2. J. Borgström, A.D. Gordon, M. Greenberg, J. Margetson, and J. Van Gael. Measure Transformer Semantics for Bayesian Machine Learning. In *Programming Languages and Systems*, 77-96, Springer Berlin Heidelberg, 2011.
3. P. Martin Löf. An Intuitionistic Theory of Types: Predicative Part. In *Logic Colloquim, 1973*, North-Holland, 1975.
4. M. Richardson, and P. Domingos. Markov Logic Networks. *Machine Learning*, 62(1-2):107-136, 2006.
5. K.S. Ng, and J.W. Lloyd. Probabilistic Reasoning in a Classical Logic. *Journal of Applied Logic*, 7(2):218-238, 2007.
6. M. Hutter, J.W. Lloyd, K.S. Ng, and W.T.B. Uther. Probabilities on Sentences in an Expressive Logic. *Journal of Applied Logic*, 11(4):386-420, 2013.
7. R. Cooper, S. Dobnik, S. Lappin, and S. Larsson. A Probabilistic Rich Type Theory for Semantic Interpretation. *Proceedings of the EACL 2014 Workshop on Type Theory and Natural Language Semantics*, 72-79, 2014.
8. H. P. Barendregt. Lambda Calculi with Types. In *Handbook of Logic in Computer Science*, Vol 2:1-189, 1992.
9. S. Thompson. *Type Theory and Functional Programming*, Addison-Wesley, 1991.

10. P. Martin-Löf. ”Constructive mathematics and computer programming.” In *Studies in Logic and the Foundations of Mathematics*, vol. 104, pp. 153-175. Elsevier, 1982.
11. The *Univalent Foundations Program*, *Homotopy Type Theory: Univalent Foundations of Mathematics*, Princeton NJ, Institute for Advanced Study, 2013, https://homotopytypetheory.org/book.
12. C. Kaliszyk, F. Chollet, and C. Szegedy. Holstep: A machine learning dataset for higher-order logic theorem proving, *ICLR*, 2017.
13. T. Rocktäschel, and S. Riedel. End-to-end differentiable proving. *NIPS* 2017.
14. D. Tran, M. D. Hoffman, R. A. Saurous, E. Brevdo, K. Murphy, and D. M. Blei. Deep probabilistic programming, *ICLR*, 2017.
15. V. Voevodsky, B. Ahrens, and D. Grayson, *UniMath*, 2014, https://github.com/UniMath.