Exact Inference for Relational Graphical Models with Interpreted Functions: Lifted Probabilistic Inference Modulo Theories

Rodrigo de Salvo Braz   **Ciaran O'Reilly**
Artificial Intelligence Center
SRI International
Menlo Park, CA USA

Abstract

Probabilistic Inference Modulo Theories (PIMT) is a recent framework that expands exact inference on graphical models to use richer languages that include arithmetic, equalities, and inequalities on both integers and real numbers. In this paper, we expand PIMT to a *lifted* version that also processes random functions and relations. This enhancement is achieved by adapting *Inversion*, a method from Lifted First-Order Probabilistic Inference literature, to also be modulo theories. This results in the first algorithm for exact probabilistic inference that efficiently and simultaneously exploits random relations and functions, arithmetic, equalities and inequalities.

1 INTRODUCTION

Graphical models such as Bayesian networks and Markov networks (Pearl, 1988) are a well-principled way of representing probabilistic models. They represent the dependences between random variables in a factorized form that can be exploited by probabilistic inference algorithms (such as Variable Elimination (Zhang and Poole, 1994) and Belief Propagation (Pearl, 1988)) for greater efficiency.

However, traditional graphical models representations are unable to represent other types of useful structures:

- **Relational structure** occurs when families of random variables share the same dependences. Suppose a model involves random variables $sunny$, $happy(Ann)$, $happy(Bob), \dots$ and the analogous dependences $P(happy(Ann)|sunny)$, $P(happy(Bob)|sunny)$ and so on. Intuitively, this structure can be exploited for greater efficiency since the same inference can often be performed only once for an entire family of random variables. Traditional graphical model representations cannot explicitly indicate this structure and thus algorithms cannot directly exploit it.

- **Algebraic structure** occurs when dependences between random variables (such as conditional probabilities) can be compactly described by an algebraic expression. For example, it may be that $P(x \in \{1, \dots, 1000\}|y \in \{1, \dots, 1000\})$ is equal to `if` $x = y$ `then` 0.8 `else` $0.2/999$. Intuitively, this can be exploited for greater efficiency because large groups of values can be treated in the same way (in this case, all pairs of values (x, y) for which $x \neq y$ are equally probable). Traditional graphical model representations, however, represent such functions as a lookup table (a large 1000×1000 one for this example)[1], thus not explicitly representing the algebraic structure, which prevents algorithms from directly exploiting it.

These types of structure are commonly found in real-world applications and fields such as probabilistic programming, so research has been conducted on exploiting both:

- **Lifted probabilistic inference** (Poole, 2003; de Salvo Braz, 2007; Milch *et al.*, 2008; Van den Broeck *et al.*, 2011; Kersting, 2012) exploits **relational structure** that is explicitly represented with richer languages generally known as **Probabilistic Relational Models**. These representations resemble universal quantification in first-order logic such as $\forall x \in People \, P(happy(x)|sunny)$ for $People$ a discrete set $\{Ann, Bob, \dots\}$. In logic terminology, this employs **uninterpreted functions**, since the random relations are Boolean functions without a fixed interpretation (for example, the value of $happy(Ann)$ is not fixed in advance).

- **Probabilistic Inference Modulo Theories (PIMT)** (de Salvo Braz *et al.*, 2016) exploits **algebraic structure** by explicitly representing and manipulating function definitions of random variable dependences in the form of algebraic expressions. In logic terminology, this employs **interpreted functions**, because functions like equality

---

[1] But note exceptions in Boutilier *et al.* (1996); Sanner and Abbasnejad (2012).

(=) and arithmetic functions (+, × etc), among others, have *fixed* interpretations: $3 = 3$ is always TRUE, and $1 + 3$ is always 4.

In this paper, we present the **first lifted probabilistic algorithm on languages with interpreted functions**. This is done by **unifying these two lines of research** by incorporating uninterpreted functions and an important lifted probabilistic inference operation, **Inversion**, into PIMT. (Another major lifted operation, Counting (de Salvo Braz, 2007; Milch *et al.*, 2008), is left for future work.) We call this fusion of lifted inference and PIMT **Lifted Probabilistic Inference Modulo Theories (LPIMT)**.

Moreover, we achieve this unification *modularly*, by using PIMT's ability to be extended by solvers for specific theories, and by encapsulating lifted inference into it as an *uninterpreted functions theory solver*. This means the algorithm can apply lifted inference even to unanticipated theories, because solvers for different theories are orthogonally applied by the general, theory-agnostic level of PIMT.

Casting lifted inference in the PIMT framework also makes it simpler and more powerful than previous lifted inference methods. Besides the above mentioned advantages, it also uses standard mathematical notation (e.g., not making a hard distinction between "logical" and "random" variables); does not separate constraints from potential function definitions; and accepts random function applications even as arguments to other functions (nested applications).

2 BACKGROUND

2.1 Graphical Models and Variable Elimination

Graphical models are a standard framework for reasoning with uncertainty. The most common types are Bayesian networks and Markov networks. In both cases, a **joint probability distribution** for each assignment tuple $\mathbf{x}$ to N random variables is defined as a normalized product of non-negative real functions $\{\phi_i\}_{i \in 1..M}$, where $1..M$ is short for $\{1, \ldots, M\}$, each of them applied to a subtuple $\mathbf{x}_i$ of $\mathbf{x}$:[2]

$$P(\mathbf{x}) = \frac{1}{Z} \prod_i \phi_i(\mathbf{x}_i),$$

where Z is a normalization constant equal to $\sum_{\mathbf{x}} \prod_i \phi_i(\mathbf{x}_i)$. Functions ϕ_i are called **factors** and map each assignment on their arguments to a **potential**, a non-negative real number that represents how likely the assignment $\mathbf{x}_i$ is. This representation is called **factorized** due to its breaking the joint probability into this product. In Bayesian networks, $M = N$ and factors are conditional probabilities $P(x_i|Pa_i)$, for each random variable x_i in $\mathbf{x}$, where Pa_i are its parents in a directed acyclic graph.

---

[2]For simplicity, we use the same symbols for both random variables and their values, but the meaning should be clear.

The **marginal probability (MAR)** problem consists of computing $P(\mathbf{q}) = \sum_{\mathbf{r}} P(\mathbf{x})$, where $\mathbf{q}$ is a subtuple of $\mathbf{x}$ containing queried variables, and $\mathbf{r}$ are all the remaining variables in $\mathbf{x}$. It can be shown that $P(\mathbf{q}) = \frac{1}{Z_{\mathbf{q}}} \sum_{\mathbf{r}} \prod_i \phi_i(\mathbf{x}_i)$ for $Z_{\mathbf{q}}$ a normalization constant over $\mathbf{q}$. Therefore, the problem can be easily reduced to computing a **summation over products of factors**, which the rest of the paper focuses on.

The **belief update (BEL)** problem consists of computing $P(\mathbf{q}|\mathbf{e})$, the posterior probability on variables $\mathbf{q}$ given assignment $\mathbf{e}$ to evidence variables (an observation). BEL can be easily reduced to two instances of MAR, since Bayes' theorem establishes that $P(\mathbf{q}|\mathbf{e}) = P(\mathbf{q}, \mathbf{e})/P(\mathbf{e})$, where $P(\mathbf{q}, \mathbf{e})$ is the joint value of $\mathbf{q}$ and $\mathbf{e}$.

Computing $\sum_{\mathbf{r}} \prod_i \phi_i(\mathbf{x}_i)$ is crucial for inference with graphical models, but solving it naively has exponential cost in the size of $\mathbf{r}$. **Variable Elimination (VE)** is an algorithm that takes advantage of the factorized representation to more efficiently compute this sum. For example, consider the following summation given a factorization on the variables (h)appy, (w)eekday, (t)emperature, (m)onth with range sizes of 2 (true or false), 7 (weekdays), 3 ($hot, mild, cold$), and 12 (months) respectively:

$$\sum_{h,w,t,m} P(h|w,t)P(w)P(t|m)P(m).$$

A naive computation iterates over 504 assignments to (h, w, t, m). However, the joint probability factorization enables us to manipulate the expression and compute it with iterations over fewer assignments:

$$\sum_{h,w,t,m} P(h|w,t)P(w)P(t|m)P(m)$$
$$= \sum_{h,w,t} P(h|w,t)P(w) \sum_m P(t|m)P(m).$$

Now, we **sum** m **out** (or **eliminate** it), obtaining a new factor ϕ, defined on t alone, that replaces the last summation. This requires going over each value t and computing $\phi(t) = \sum_m P(t|m)P(m)$ (thus iterating over the values of m). Therefore, computing ϕ takes iterating over 36 assignments to t, m and we are left with the following new summation of a product of functions:

$$\sum_{h,w,t} P(h|w,t)P(w)\phi(t)$$
$$= \sum_w P(w) \sum_{h,t} P(h|w,t)\phi(t)$$
$$= \sum_w P(w)\phi'(w) \text{ (after 42 assignments to } w, h, t)$$
$$= \phi'' \text{ (after 7 assignments to } w).$$

Variable Elimination therefore decreases the number of required iterations to 85, a six-fold reduction, by exploiting

the fact that not all variables share factors with all variables (for example, m only shares a factor with t).

VE applies not only to summations of products, but to any commutative associative semiring (Bistarelli *et al.*, 1997): maximization of products, disjunctions of conjunctions, conjunctions of disjunctions, and so on. We use $\oplus$ and $\otimes$ for the **additive and multiplicative operators** of the semiring. We call **quantifiers** the intensional versions of operators: $\forall$ for $\wedge$, $\exists$ for $\vee$, $\sum$ and $\int$ for $+$ (for discrete and continuous indices respectively), $\prod$ for $\times$, Max for max, and so on. Quantifiers corresponding to $\oplus$ and $\otimes$ are denoted $\bigoplus$ and $\bigotimes$, respectively. We use $\bigodot$ for denoting any type of quantifier. VE works by selecting the next variable v in $\mathbf{r}$ to be eliminated and computing a new factor

$$\phi(\mathbf{u}) = \bigoplus_{v} \bigotimes_{i \in \mathbf{f}(v)} \phi_i(\dots, v, \dots),$$

where $\mathbf{f}(v)$ is the set of indices of factors of v and $\mathbf{u}$ are the variables sharing factors with v. It then includes ϕ in the product of factors and proceeds until all variables in $\mathbf{r}$ are eliminated.

2.2 Probabilistic Inference Modulo Theories

Typically, graphical model inference algorithms assume that factors can only be accessed as opaque lookup tables, that is, by providing an assignment to its random variables. This requires Variable Elimination to iterate over all assignments to all random variables in a summation, as seen in the example for computing $\phi(t)$ with Variable Elimination.

However, often the *definitions* of factors are available in the form of **symbolic algebraic expressions** that use operators from specific theories. For example, the conditional probability of temperature given month, and the prior probability of month in our original example could be represented by[3]

$$P(t|m) = \texttt{if } m \le 3$$
$$\texttt{then if } t = cold \texttt{ then } 0.8 \texttt{ else } 0.1$$
$$\texttt{else if } t = cold \texttt{ then } 0.4 \texttt{ else } 0.3$$
$$P(m) = 1/12.$$

While this form does not preclude regular VE to access it as a lookup table (by computing its result given an assignment to t and m), the symbolic expression is itself available as a data structure. This makes the structure of the factor evident; for example, specific values of m do not matter, but only whether $m \le 3$.

Probabilistic Inference Modulo Theories (PIMT), exploits this available symbolic information for much faster inference. It does so by using **Symbolic Evaluation Modulo Theories (SEMT)**, which is a method for simplifying

---

[3] Note that $P(t|m)$ sums up to 1 for each m because *hot* and *mild* both have the `else` case probability mass.

(and, when possible, completely evaluating) symbolic algebraic expressions, including eliminating quantifiers such as $\sum, \prod, \exists$, and $\forall$. SEMT applies to general expressions, and not only probabilistic reasoning-related ones. PIMT is simply the use of SEMT applied to expressions that happen to be (unnormalized) marginal probabilities, so we mostly refer to SEMT from now on, keeping in mind that it includes PIMT as a special case.

SEMT and PIMT are similar to **Satisfiability Modulo Theories (SMT)** solvers (Barrett *et al.*, 2009; de Moura *et al.*, 2007), which also take modular theory-specific solvers to solve multi-theory problems. However, they generalize SMT in two important ways: they are **quantifier-parametric**, that is, they eliminate multiple types of quantifiers and deal with expressions of any type, including Boolean, numeric and categorical, as opposed to SMT only solving existential quantification on Boolean-valued formulas; and they are **symbolic**, that is, can process free variables and return results expressed in terms of them, as opposed to assuming all variables to be existentially quantified as in SMT.

SEMT is a variant of SGDPLL(T) (de Salvo Braz *et al.*, 2016) slightly generalized to deal with any expression, and not just sequences of nested quantifiers, and to make use of *contexts*, defined below. We give an overview of SEMT here but refer to the SGDPLL(T) reference for details.

SEMT receives a pair (E, C) as input. E is the expression being evaluated, and C the **context**. Let $\mathbf{x}$ be the free variables in E. Context C is a formula on $\mathbf{x}$ that indicates that *only* the assignments on $\mathbf{x}$ satisfying C need be considered. For example, the expression $\texttt{if } x \ne 1 \texttt{ then } 2 \texttt{ else } 4$, under context $x = 2 \vee x = 3$, can be safely evaluated to 4 because $x \ne 1$ is never true under that context.

SEMT traverses the expression top-down and evaluates each sub-expression according to a set of **rewriting rules** (presented below) until no rule applies to any sub-expressions. It also keeps track of the context holding for sub-expression, depending on the expressions above it. For example, in $\texttt{if } x = 1 \texttt{ then } \sum_{i \in 1..10: i \ne 3} \gamma \texttt{ else } \varphi$, γ is evaluated under context $x = 1 \wedge i \ne 3$.

Let $E[\alpha/\beta]$ be the **substitution** of all occurrences of expression α in expression E by β. The SEMT rewriting rules are the following (an example is provided afterwards):

- **simplification**: simplifiers for each theory are applied when possible: examples are $1 + 2 \to 3$, $x = x \to true$, $\texttt{if } true \texttt{ then } 1 \texttt{ else } 2 \to 1$, $0 \times x \to 0$ and so on. These always decrease the size of the expression.

- **literal determination**: if the expression is a literal L under a context C and $\forall V : C \Rightarrow L$, where V are the free variables in C and L, rewrite the expression to TRUE; if $\forall V : C \Rightarrow \neg L$, rewrite it to FALSE; the tautology may be decided by SEMT itself or an SMT solver;

- **factoring out**: if ϕ_1 does not involve index i, $\bigodot_{i \in D:C}(\phi_1 \odot \phi_2) \to \phi_1 \odot \bigodot_{i \in D:C} \phi_2$.

- **if-splitting**: if the expression is ϕ and contains a literal L undefined by the context C, rewrite it to if L then $\phi[L/\text{TRUE}]$ else $\phi[L/\text{FALSE}]$.

- **quantifier-splitting**: rewrite expressions of the form $\bigodot_{i \in D:C} \phi$, where i is a variable of type D, F is a conjunction of literals in the theory for type D, and ϕ is expression containing a literal L containing i, to a new expression containing two quantified expressions, each with one less literal in its body: $\left(\bigodot_{i:F \wedge L} \phi[L/\text{TRUE}] \right) \odot \left(\bigodot_{i:F \wedge \neg L} \phi[L/\text{FALSE}] \right)$.

- **theory-specific quantifier elimination**: if ϕ does not contain any literals, solve $\bigodot_{i \in D:F} \phi$ with a provided, modular *theory-specific* solver for the type of i.

SEMT always returns a quantifier-free expression since the theory-specific quantifier elimination is eventually invoked for each quantifier. For an expression representing a marginal over a product of factors, it reproduces Variable Elimination by summing out one variable at a time, with the advantage of exploiting factors represented symbolically.[4]

Consider the computation of $\phi(t) = \sum_m P(t|m)P(m)$ below. Regular VE requires iterating over 36 assignments to t, m, but SEMT only needs to consider the 4 truth assignments for literals $t = cold$ and $m \leq 3$:

$$\sum_m (\text{if } m \leq 3 \text{ then if } t = cold \text{ then } 0.8 \text{ else } 0.1$$

$$\text{else if } t = cold \text{ then } 0.4 \text{ else } 0.3)/12$$

$\to$ (by if-splitting on $t = cold$ and simplification)

if $t = cold$

then $\sum_m (\text{if } m \leq 3 \text{ then } 0.8 \text{ else } 0.4)/12$

else $\sum_m (\text{if } m \leq 3 \text{ then } 0.1 \text{ else } 0.3)/12$

$\to$ (by quantifier-splitting on $m \leq 3$ in first summation)

if $t = cold$

then $\sum_{m:m \leq 3} (\text{if TRUE then } 0.8 \text{ else } 0.4)/12$

$+ \sum_{m:\neg(m \leq 3)} (\text{if FALSE then } 0.8 \text{ else } 0.4)/12$

else $\sum_m (\text{if } m \leq 3 \text{ then } 0.1 \text{ else } 0.3)/12$

$\to$ (by simplification and quantifier-splitting on $m \leq 3$)

if $t = cold$ then $0.8/12 \times 3 + 0.4/12 \times 9$

else $0.1/12 \times 3 + 0.3/12 \times 9$

$\to$ if $t = cold$ then 0.5 else 0.25,

which represents the resulting new factor $\phi(t)$ (which happens to be $P(t)$) to be used in the next steps of VE. Note that the expressions above are *not* just a mathematical argument, but the actual data structures manipulated by SEMT.

SEMT (and consequently PIMT) is **modulo theories** because symbolic evaluation is **theory-agnostic**, working for any theory T given an encapsulated **theory solver** that provides quantifier elimination solvers and simplification rules for T. Theory solvers must be provided only for the simpler expressions of the type $\bigodot_{i \in D:F} \phi$ where ϕ is literal-free and F is a conjunction of literals in the theory. de Salvo Braz *et al.* (2016) details a solver for summation over polynomials, with literals in difference arithmetic on bounded integers. A similar solver for integrals over polynomials and linear real arithmetic literals has also been defined since. An example of SEMT on multiple theories with only 4 cases is:

$$\sum_{i \in 1..7} \int_{x \in [0;10]} \text{if } i \geq 3 \text{ then if } x < 5 \text{ then } x^2 \text{ else } 0 \text{ else } i$$

$$\to \sum_{i \in 1..7} \text{if } i \geq 3 \text{ then } \int_{x \in [0;10]} \text{if } x < 5 \text{ then } x^2 \text{ else } 0$$

$$\text{else } \int_{x \in [0;10]} i$$

$$\to \sum_{i \in 1..7} \text{if } i \geq 3 \text{ then } \int_{x \in [0;10]:x<5} x^2 + \int_{x \in [0;10]:x \geq 5} 0$$

$$\text{else } 10i$$

$$\to \sum_{i \in 1..7} \text{if } i \geq 3 \text{ then } 125/3 \text{ else } 10i$$

$$\to \left(\sum_{i \in 1..7:i \geq 3} 125/3 \right) + \left(\sum_{i \in 1..7:i<3} 10i \right)$$

$$\to 625/3 + 30 \to 715/3.$$

2.3 Relational Graphical Models

Relational graphical models (RGM) specify probability distributions over families of random variables that can be seen as relations. There are many RGM languages in the literature; they vary significantly in syntax and superficial features, but are typically equivalent and convertible to one another. Markov Logic Networks (MLN) (Richardson and Domingos, 2004) is a well-known one. It consists of a set of weight formulas, an example of which is

$2.5 : Smoker(Bob)$

$1.4 : Smoker(X) \wedge Friends(X, Y) \Rightarrow Smoker(Y)$

for **logical** variables X and Y ranging over a finite set $D = \{Ann, Bob, Charlie, \dots\}$.

The random variables in this MLN are the **groundings**, or **instantiations**, of the relations in it for every assignment to the logical variables: $Sm(Ann)$, $Sm(Bob)$, ..., $Fr(Ann, Ann)$, $Fr(Ann, Bob)$, ... (we abbreviate $Smoker$ and $Friends$ from now on). A formula with weight w defines a factor for each of its instantiations (one for each assignments to its logical variables). The potential assigned by such factors is e^w if the formula is true, and 1 otherwise. Therefore, some of the factors of this MLN are: $\phi_1(Sm(Bob))$, $\phi_2(Sm(Ann), Fr(Ann, Bob), Sm(Bob))$, $\phi_2(Sm(Ann), Fr(Ann, Charlie), Sm(Charlie))$, and $\phi_2(Sm(Bob), Fr(Bob, Ann), Sm(Ann))$, where ϕ_1 and ϕ_2 are potential functions applied to all factors instantiated from the first and second formulas respectively:

$$\phi_1(s) = \texttt{if } s \texttt{ then } e^{2.5} \texttt{ else } 1$$

$$\phi_2(s_1, f, s_2) = \texttt{if } s_1 \wedge f \Rightarrow s_2 \texttt{ then } e^{1.4} \texttt{ else } 1.$$

Because the number of instantiations can be huge, performing inference on RGMs by simply instantiating them as regular graphical models is often infeasible. The next section describes **Inversion**, one of the operations used in **lifted probabilistic inference**, which exponentially improves efficiency in RGMs in many cases.

3 INVERSION AND INVERSION MODULO THEORIES

This section presents this paper's main contributions. We present a new formulation for RGMs and Inversion that is more algebraically standard, and then use this formulation to generalize Inversion to Inversion Modulo Theories.

3.1 RGMS with Function-Valued Random Variables

While the RGM literature considers $Sm(Ann)$, $Sm(Bob), \dots$ as individual (Boolean) random variables, it is conceptually simpler, and more mathematically standard, to talk about Sm as a single random variable that happens to be *function-valued*. From this point of view, this MLN has only two random variables: Sm and Fr, and defines the following joint probability distribution:

$$P(Sm, Fr) = \frac{1}{Z} \times \phi_1(Sm(Bob))$$
$$\times \prod_{X \in D} \prod_{Y \in D} \phi_2(Sm(X), Fr(X, Y), Sm(Y))$$

where Z is defined as usual, as well as marginalization:

$$P(Sm) = \sum_{Fr \in D \times D \to Boolean} P(Sm, Fr).$$

Note that, given the form of $P(Sm, Fr)$, marginal probabilities in RGMs are **sums of products of factors**, including *intensional* products using $\prod$. This fact is heavily exploited by Inversion.

To compute the marginal of a specific function application, say, $Sm(Ann)$, we need to marginalize over all *remaining* random variables, that is, Fr and all $Sm(x)$ for $x \in D \setminus \{Ann\}$. This requires splitting the variable Sm into two distinct function-valued random variables: $Sm' : \{Ann\} \to Boolean$ and $Sm'' : D \setminus \{Ann\} \to Boolean$, and replace each application $Sm(\theta)$ of the original Sm on argument expression θ by
$\texttt{if } \theta = Ann \texttt{ then } Sm'(Ann) \texttt{ else } Sm''(\theta)$. Then we can compute

$$P(Sm') = \sum_{Sm'':D\setminus\{Ann\}\to Boolean} \sum_{Fr} P(Sm', Sm'', Fr).$$

The above shows that solving RGMs is simply equivalent to allowing function-valued random variables in the model. However, summations over functions are expensive due to their high number of possible values ($2^{|D|^2}$ for Fr, for instance). The next section describes a method that exponentially decreases this cost in some cases.

3.2 Inversion on Function-Valued Variables

Lifted probabilistic inference algorithms seek to exploit the structure of random functions for greater efficiency. It includes a few operations, but in this paper we consider only one: **Inversion** (also called Lifted Decomposition).

Inversion uses the fact that summations indexed by functions of products of factors may under certain conditions be transformed into exponentially cheaper summations over "slices" of the original function. Its name comes from the fact that a summation of products becomes a cheaper product of a summation ($\sum \prod \to \prod \sum$), thus "inverting" the quantifiers. Consider an example in which the body ϕ of the sum-product depends on a single application of a function f ranging over the set of functions $1..10 \to 1..5$:

$$\sum_{f \in (1..10 \to 1..5)} \prod_{x \in 1..10} \phi(f(x)) = \prod_{x \in 1..10} \sum_{f \in (\{x\} \to 1..5)} \phi(f(x)).$$

Note that, while f ranges over $(1..10 \to 1..5)$ on the left-hand side, and thus over 5^{10} possible values, the domain of f on the right-hand side is a singleton set, because x is bound by the now outer $\prod_x$. This reduces the number of possible values of f to only 5, making iterating over it exponentially cheaper.

The equality holds because x "slices" f into independent portions:

$$\sum_{f \in (1..10 \to 1..5)} \prod_{x \in 1..10} \phi(f(x))$$
$$= \sum_{f_1 \in \{1\} \to 1..5} \cdots \sum_{f_{10} \in \{10\} \to 1..5} \phi(f_1(1)) \times ... \times \phi(f_{10}(10))$$
$$= \sum_{f_1 \in \{1\} \to 1..5} \phi(f_1(1)) \cdots \sum_{f_{10} \in \{10\} \to 1..5} \phi(f_{10}(10)) \quad (*)$$

$$= \left(\sum_{f_1 \in \{1\} \to 1..5} \phi(f_1(1)) \right) \dots \left(\sum_{f_{10} \in \{10\} \to 1..5} \phi(f_{10}(10)) \right)$$

$$= \left(\sum_{f \in \{1\} \to 1..5} \phi(f(1)) \right) \dots \left(\sum_{f \in \{10\} \to 1..5} \phi(f(10)) \right)$$

$$= \prod_{x \in 1..10} \sum_{f \in (\{x\} \to 1..5)} \phi(f(x)).$$

Once transformed in this way, we proceed as follows:

$$\prod_{x \in 1..10} \phi(1) + \phi(2) + \phi(3) + \phi(4) + \phi(5)$$

$$= \prod_{x \in 1..10} \phi' \quad = \quad (\phi')^{10} \quad = \quad \phi''$$

by using the fact that constant ϕ' does not depend on x.

The above transformation is valid because $\prod_x \phi(f(x))$ contains 10 factors, each of them only involving the application of f to a *single* value of x. This means they can be factored out of the summations indexed by other applications of the function, resulting in smaller and equivalent summations that are computed only once and then exponentiated. Similar transformations may be applied even if there are products over more than one variable:

$$\sum_{f \in A_1 \times A_2 \to B} \prod_x \prod_y \phi(f(x,y)) = \prod_x \prod_y \sum_{f \in \{(x,y)\} \to B} \phi(f(x,y))$$

by an analogous argument.

However, the transformation is not always valid:

$$\sum_{f \in A_1 \times A_2 \to B} \prod_x \prod_y \phi(f(x,y), f(y,x))$$

$$\neq \prod_x \prod_y \sum_{f \in \{(x,y)\} \to B} \phi(f(x,y), f(y,x)).$$

To see this, consider a pair $(a,b) \in A_1 \times A_2$. The summation $\sum_{f \in \{(a,b)\} \to B} \phi(f(a,b), f(b,a))$ depends on $f(a,b)$ and $f(b,a)$, which both occur in $\sum_{f \in \{(b,a)\} \to B} \phi(f(b,a), f(a,b))$, and this shared dependence prevents factoring as in step (*) above.

This does not mean that having more than one application of f in ϕ admits no Inversion, but it may restrict the inversion to just *some* of the products:

$$\sum_{f \in ((A_1 \times A_2 \times A_3) \to B)} \prod_x \prod_y \phi(f(x,y,x), f(x,3,x)) \quad (1)$$

$$= \prod_x \sum_{f \in ((\{x\} \times A_2 \times \{x\}) \to B)} \prod_y \phi(f(x,y,x), f(x,3,x)),$$

which does not decrease the function's domain size to 1 but still exponentially decreases the evaluation cost.

All these cases are covered by the following theorem:

Theorem 3.1 (Inversion). *Let E be an expression in which all applications of a function f have their first k arguments (without loss of generality because they can be permutated) equal to $x_{i_1}, \dots, x_{i_k}$ for i a k-tuple of indices in $\{1, \dots, m\}$. If operator $\oplus$ distributes over operator $\otimes$,*

$$\bigoplus_{f \in (A_1 \times \dots \times A_n) \to B} \bigotimes_{x_1} \dots \bigotimes_{x_m} E$$

$$= \bigotimes_{x_1} \dots \bigotimes_{x_m} \bigoplus_{f \in (\{(x_{i_1}, \dots, x_{i_k})\} \times A_{k+1} \times \dots \times A_n) \to B} E,$$

which is exponentially cheaper to evaluate.

Note that E may contain quantifiers itself; this was the case in Equation (1). The theorem's proof mirrors the operations shown in the examples above.

Crucially, Theorem 3.1 uses a *syntactic* check that relies on the language allowing only simple terms (variables and constants) as arguments to functions. It therefore does not apply to the important extensions in which more complex terms, using theory-specific operators such as arithmetic or inequalities, or even operators from unanticipated theories, are used as function arguments. We will see how this can be done with a *semantic* check in the next section.

3.3 Inversion Modulo Theories

We now present this paper's main contribution: **Inversion Modulo Theories**, which is a version of the lifted probabilistic inference operation Inversion in the presence of factors defined through symbolic algebraic expressions involving multiple theories. Like regular Inversion, Inversion Modulo Theories does not cover every possible problem, in which case one must fall back to currently used methods like grounding or sampling. Future work includes generalizing other lifted inference methods like Counting (de Salvo Braz, 2007; Milch *et al.*, 2008) to also be modulo theories, thus decreasing the need for the fallback methods.

Previous work on Inversion assumed RGMs expressed in a simple language in which the only allowed arguments in function applications are variable or constant symbols. This enables the Inversion condition to consist of a *syntactic* check. However, once we allow function arguments to be arbitrary terms from any of the available theories, these syntactic tests no longer apply, since the arguments semantics depends on the particular theory and the syntactic check does not take semantics into account. The new Inversion test must apply to arguments from any theory and, in fact, to arguments in even *new, as of yet unanticipated* theories. This requires this new test to be theory-agnostic, and defined in such a way that theory solvers may be transparently employed.

We start by defining $oc_f[E]$, an expression representing the portion of the domain of a given function f involved in a given expression E.

Definition 3.1. *The **set of argument tuples for** $f : A \to B$ **occurring in an expression** E is denoted $oc_f[E]$ and inductively defined as follows:*

- *if E does not contain f, $oc_f[E]$ is $\emptyset$;*

- *if E is $f(t)$ for t a tuple, $oc_f[E]$ is $\{t\}$;*

- *if E is f, $oc_f[E]$ is A;*

- *if E is $\texttt{if } C \texttt{ then } E_1 \texttt{ else } E_2$ and C does not contain f, then $oc_f[E]$ is $\texttt{if } C \texttt{ then } oc_f[E_1] \texttt{ else } oc_f[E_2]$; otherwise, $oc_f[E]$ is $oc_f[C] \cup oc_f[E_1] \cup oc_f[E_2]$;*

- *if E is $g(t_1, \dots, t_k)$ for g a function symbol distinct from f, or E is $\{t_1, \dots, t_k\}$, $oc_f[E]$ is $oc_f[t_1] \cup \cdots \cup oc_f[t_k]$;*

- *if E is $Q_{x \in T:C} E'$ for Q an arbitrary quantifier,*

    - *if C does not contain f, then $oc_f[E]$ is $oc_f[T] \cup \bigcup_{x \in T:C} oc_f[E']$;*
    - *otherwise, $oc_f[E]$ is $oc_f[T] \cup \bigcup_{x \in T}(oc_f[C] \cup oc_f[E'])$.*

For example,

$$oc_f\big[f(x,y) + \sum_{z \in 1..10:z!=3} f(x,z) \prod_{w \in 1..10} f(w,z)\big]$$
$$= \{(x,y)\} \cup \bigcup_{z \in 1..10:z!=3} \big(\{(x,z)\} \cup \bigcup_{w \in 1..10} \{(w,z)\}\big),$$

in which x and y are free variables.

As a further preliminary step, we add simplification rules for tuples, and for testing whether sets are empty, to be used by SEMT. This will be useful later in manipulating $oc_f[E]$ expressions.

Theorem 3.2 (Tuple and Empty Set Simplifiers). *The following **tuple and empty set simplifiers***

1. *$(r_1, \dots, r_n) = (s_1, \dots, s_n) \to$ $r_1 = s_1 \wedge \cdots \wedge r_m = s_m$ (or its negation for $\neq$).*

2. *$t \in \{t_1, \dots, t_n\} \to (t = t_1) \vee t \in \{t_2, \dots, t_n\}$.*

3. *$t \in \bigcup_{i \in D:C} \phi \to \exists i \in D : (C \wedge t \in \phi)$.*

4. *$\big(\bigcup_{i \in D:C} \phi\big) = \emptyset \to \forall i \in D : (\neg C \vee \phi = \emptyset)$.*

5. *$S \cap \emptyset = \emptyset \to \text{TRUE}$.*

6. *$S \cup \emptyset = \emptyset \to S = \emptyset$.*

7. *$S \cap \{t_1, \dots, t_n\} = \emptyset \to$ $(t_1 \notin S) \wedge (S \cap \{t_2, \dots, t_n\} = \emptyset)$.*

8. *$\big(\bigcup_{i \in D:C} \phi\big) \cap \big(\bigcup_{i' \in D':C'} \phi'\big) = \emptyset \to$ $\forall i \in D : C \Rightarrow \forall i' \in D' : C' \Rightarrow (\phi \cap \phi' = \emptyset)$.*

9. *$(S_1 \cup S_2) \cap S_3 = \emptyset \to (S_1 \cap S_3) \cup (S_2 \cap S_3) = \emptyset$.*

10. *$S_1 \cup S_2 = \emptyset \to S_1 = \emptyset \wedge S_2 = \emptyset$.*

11. *$\{t_1, \dots, t_n\} = \emptyset \to \text{FALSE}$ if $n > 0$, TRUE otherwise,*

when included in SEMT, rewrite $oc_f[E] = \emptyset$ expressions to equivalent formulas free of tuple and set expressions.

The proof is presented in Supplementary Materials. It is based on gradual distribution of $\cap$ over $\cup$, and conversion of intersections to comparisons between set elements:

$$\bigcup_{x \in 1..5} \big(\{(x,y)\} \cup \bigcup_{z \in 3..5} \{(z,3)\}\big) \cap \bigcup_{w \in 1..10} \{(1,w)\} = \emptyset$$

$\to$ (rule 8)

$\forall x \in 1..5 : \forall w \in 1..10 :$
$$\big(\{(x,y)\} \cup \bigcup_{z \in 3..5} \{(z,3)\}\big) \cap \{(1,w)\} = \emptyset$$

$\to$ (rule 9 distributes $\cap$ over $\cup$)

$\forall x \in 1..5 : \forall w \in 1..10 :$
$$\{(x,y)\} \cap \{(1,w)\} \cup \big(\bigcup_{z \in 3..5} \{(z,3)\}\big) \cap \{(1,w)\} = \emptyset$$

$\to$ (rule 10)

$\forall x \in 1..5 : \forall w \in 1..10 :$
$$\{(x,y)\} \cap \{(1,w)\} = \emptyset \wedge \big(\bigcup_{z \in 3..5} \{(z,3)\}\big) \cap \{(1,w)\} = \emptyset$$

$\to$ (rules 7, 2, and 8)

$\forall x \in 1..5 : \forall w \in 1..10 :$
$\quad (x,y) \neq (1,w) \wedge \forall z \in 3..5 : \{(z,3)\} \cap \{(1,w)\} = \emptyset$

$\to$ (rule 1 on first conjunct, rules 7 and 2 on second one)

$\forall x \in 1..5 : \forall w \in 1..10 :$
$\quad (x \neq 1 \vee y \neq w) \wedge \forall z \in 3..5 : (z,3) \neq (1,w)$

$\to$ (rule 1 for breaking tuples)

$\forall x \in 1..5 : \forall w \in 1..10 :$
$\quad (x \neq 1 \vee y \neq w) \wedge \forall z \in 3..5 : z \neq 1 \vee 3 \neq w$

$\to$ (integer-specific solver for $\bigodot = \forall$ for z, w and x)

$\forall x \in 1..5 : \forall w \in 1..10 : (x \neq 1 \vee y \neq w) \wedge 3 \neq w$

$\to \forall x \in 1..5 : \text{FALSE} \quad \to \quad \text{FALSE}.$

Deciding whether a set expression is equivalent to the empty set is crucial for Inversion Modulo Theories, which can now be formally stated:

Theorem 3.3 (Inversion Modulo Theories). *Let E be an expression and T_i, C_i be type and constraint, respectively, in a theory for which we have a satisfiability solver. Then,*

$$\bigoplus_{f \in A \to B} \bigotimes_{x_1 \in T_1:C_1} \cdots \bigotimes_{x_k \in T_k:C_k} E,$$

where $A = oc_f[\bigotimes_{x_1 \in T_1:C_1} \cdots \bigotimes_{x_k \in T_k:C_k} E]$ (this is relaxed in Section 3.4), is equivalent to , and therefore can

be rewritten as,

$$\bigotimes_{x_1 \in T_1 : C_1} \cdots \bigotimes_{x_k \in T_k : C_k} \bigoplus_{f \in oc_f[E] \to B} E,$$

if $\bigotimes$ distributes over $\bigoplus$ and

$$\forall x_1' \in T_1 \ldots \forall x_k' \in T_k \quad \forall x_1'' \in T_1 \ldots \forall x_k'' \in T_k$$
$$\begin{pmatrix} C_1[x_1/x_1'] \wedge \cdots \wedge C_k[x_k/x_k'] \\ \wedge \ C_1[x_1/x_1''] \wedge \cdots \wedge C_k[x_k/x_k''] \\ \wedge \ (x_1', \ldots, x_k') \neq (x_1'', \ldots, x_k'') \end{pmatrix}$$
$$\Rightarrow \begin{pmatrix} oc_f[E][x_1/x_1', \ldots, x_k/x_k'] \\ \cap \\ oc_f[E][x_1/x_1'', \ldots, x_k/x_k''] \end{pmatrix} = \emptyset \qquad (2)$$

*(that is, the set of argument tuples in applications of f in E for different value assignments to $x_1, \ldots, x_k$ are always disjoint — we refer to this as **Condition (2)** from now on). Moreover, the rewritten expression is exponentially $(O(2^{\prod_i |\{x_i \in T_i : C_i\}|}))$ cheaper to evaluate than the original.*

The theorem (proven in Supplementary Materials) covers significantly more general cases than the original examples may suggest: f may have any arity; its arguments need not be only variables or constants, but even other (interpreted or uninterpreted) function applications; E may contain quantifiers itself; and quantifiers may involve constraints (C_i). The following example involves all these characteristics:

Example 3.4. *Let w and g be free variables.*

$$\sum_{f \in 1..10 \times (1..10 \setminus \{8\}) \times \{w+3\} \to 1..5}$$
$$\prod_{x \in (1+g(w))..(10+g(w))} \prod_{y \in 1..10 : y \neq 8}$$
$$\sum_{z \in 1..10} f(x - g(w), y, w + 3) z$$

$\to$ *(Inversion on x, y; see Eqn. (3) for Condition (2))*

$$\prod_{x \in (1+g(w))..(10+g(w))} \prod_{y \in 1..10 : y \neq 8}$$
$$\sum_{f \in \{(x-g(w), y, w+3)\} \to 1..5} \sum_{z \in 1..10} f(x - g(w), y, w + 3) z$$

$\to$ *(f has a singleton domain since x, y, g and w are fixed*
in the first summation, so it behaves like a variable v)

$$\prod_{x \in (1+g(w))..(10+g(w))} \prod_{y \in 1..10 : y \neq 8} \sum_{v \in 1..5} \sum_{z \in 1..10} vz.$$

This transformation decreases the time complexity of the first summation from the time to iterate over $5^{10 \times 9}$ values of f to 5 values of v only. But, in fact, from now on SEMT completes the calculation with no iteration at all by using

symbolic integer-specific solvers for $\sum$ and $\prod$:

$$\prod_{x \in (1+g(w))..(10+g(w))} \prod_{y \in 1..10 : y \neq 8} \sum_{v \in 1..5} \sum_{z \in 1..10} vz$$
$$\to \prod_{x \in (1+g(w))..(10+g(w))} \prod_{y \in 1..10 : y \neq 8} \sum_{v \in 1..5} 55v$$
$$\to \prod_{x \in (1+g(w))..(10+g(w))} \prod_{y \in 1..10 : y \neq 8} 55 \times 15 \to 825^{10 \times 9} \to 825^{90}.$$

In this example, Condition (2) is

$$\forall x' \in (1 + g(w))..(10 + g(w)) \quad \forall y' \in 1..10$$
$$\forall x'' \in (1 + g(w))..(10 + g(w)) \quad \forall y'' \in 1..10$$
$$(x', y') \neq (x'', y'') \wedge y' \neq 8 \wedge y'' \neq 8 \Rightarrow$$
$$\bigcup_{z \in 1..10} (x' - g(w), y', w + 3)$$
$$\cap \bigcup_{z \in 1..10} (x'' - g(w), y'', w + 3) = \emptyset \qquad (3)$$

which can be solved by SEMT with $\bigoplus$ instantiated as $\forall$ over difference arithmetic, after including tuple and empty set simplifiers from Definition 3.2).

Example 3.5. *Consider monitoring crop growth from satellite images to alert against famine. The growth (g) of crops can be determined from their color (c) in the images, and depends on whether the region was in drought (d) 3 months previously. This can be modeled as:*

$$\forall m \in Months, f \in Fields :$$
$$P(c(f, m) | g(f, m)) = \text{if } g(f, m) > 2.3 \text{ then if } c(f, m) \ldots$$
$$P(g(f, m + 3) | d(m)) = \text{if } d(m) \text{ then if } g(f, m + 3) \ldots$$

The query $P(d|c)$ requires solving the following marginalization over growth:

$$\sum_g \prod_m \prod_f P(c(f, m+3) | g(f, m+3)) P(g(f, m+3) | d(m))$$

Inversion Modulo Theories applies (because each (f, m) involves a single instance of $g(f, m + 3)$) and we obtain

$$\prod_m \prod_f \sum_{g(f, m+3)} P(c(f, m+3) | g(f, m+3)) P(g(f, m+3) | d(m))$$

which makes the summation on growth exponentially cheaper to compute and produces

$$\prod_m \prod_f \phi(c(f, m + 3), d(m)),$$

for ϕ the summation result. This can then be evaluated directly, given the evidence on color for each m and f, producing a factorized representation of the marginal on $drought(m)$, for each month. If the number of fields is 2000, and growths domain size is 5, this cuts the cost of exactly eliminating $growth$ from 5^{2000} iterations to only 5.

3.4 Dealing with Arbitrary Domains for f

Let OC_f be $oc_f[\bigotimes_{x_1 \in T_1:C_1} \cdots \bigotimes_{x_k \in T_k:C_k} E]$. Theorem 3.3 requires that $A = OC_f$, that is, that the domain of f coincide with its portion being used inside $\bigoplus_{f \in A \to B}$. When $A \neq OC_f$, we use function splitting (Section 3.1):

$$\bigoplus_{f \in A \to B} \Psi = \bigoplus_{f' \in (A \setminus OC_f) \to B} \bigoplus_{f'' \in (OC_f \setminus A) \to B} \bigoplus_{f \in (A \cap OC_f) \to B} \Psi'$$

for Ψ' obtained from Ψ after replacing each $f(\alpha)$ by

```
if α ∈ A
    then if α ∈ OC_f then f(α) else f'(α) else f''(α).
```

This technique splits the original f into three different functions f, f', f'', and replaces each of its applications by the one corresponding to its arguments. After this, the domain of f coincides with OC_f, satisfying the corresponding requirement in Theorem 3.3.

3.5 Dealing with Multiple Separate $\bigotimes$ Quantifiers

Theorem 3.3 requires a single nested sequence of $\bigotimes$ quantifiers inside $\bigoplus$. However, summations on products of *separate* $\bigotimes$ quantifiers can be rewritten to the required form:

$$\bigoplus_{f \in A \to B} \left(\left(\bigotimes_{x \in T_x:C_x} E_x \right) \left(\bigotimes_{y \in T_y:C_y} E_y \right) \right)$$
$$\to \bigoplus_{f \in A \to B} \bigotimes_{x \in T_x:C_x} \bigotimes_{y \in T_y:C_y} E_x^{\frac{1}{|\{y \in T_y:C_y\}|}} E_y^{\frac{1}{|\{x \in T_x:C_x\}|}},$$

where the exponents compensate for the extra multiplications of E_x and E_y by moving them inside $\bigotimes_y$ and $\bigotimes_x$ respectively (this is sometimes called **scaling**). Set cardinalities can be computed as a special case of summation.

In the special case in which x and y range over the same values (that is, $\{x \in T_x : C_x\} = \{y \in T_y : C_y\}$, which can be evaluated by SEMT), the two $\bigotimes$ quantifiers can be merged into a single one (or **aligned**), and the original expression is rewritten instead to the cheaper $\bigoplus_{f \in A \to B} \bigotimes_{x \in T_x:C_x} E_x E_y[y/x]$. These transformations can be easily generalized to cases with more than two $\bigotimes$ quantifications, and nested $\bigotimes$ expressions.

3.6 A Proof-of-concept Experiment

Since SEMT and Inversion Modulo Theories are elaborate symbolic algorithms, two immediate questions are whether they can be effectively implemented and how they compare with simpler alternatives such as sampling. As a proof-of-concept test, we used our implementation of SEMT for computing Example 3.4 using two alternatives: one, in which the sum-product is simplified by Inversion Modulo Theories, and another in which the sum and product

are computed by sampling. As expected, Inversion Modulo Theories vastly outperforms sampling in this example, computing the exact answer (825^{90}) in less than 300 ms, whereas sampling requires 10 minutes, and 10,000 samples per quantifier, to be within an order of magnitude of the exact answer, and 17 minutes to be within 10% error.

4 RELATED WORK AND CONCLUSION

As mentioned in the introduction, LPIMT generalizes work in the lifted inference, probabilistic inference modulo theories, and satisfiability modulo theories literatures. It is also related to Probabilistic Programs (PP) (Goodman *et al.*, 2012; Milch *et al.*, 2005), a class of high-level representations for probabilistic models that uses interpreted and uninterpreted functions. The current prevalent inference method in PPs is sampling, which is approximate and whose convergence rate depends on the size of the grounded model, that is, on the size of the domain. LPIMT can be applied to some fragments of PPs and is an exact inference alternative that avoids iterating over the domain. The closest approach to LPIMT in the PP area is Hakaru (Carette and Shan, 2016; Narayanan *et al.*, 2016), which employs symbolic methods for simplification and integration of PPs, but does not include lifted inference on random functions. Model Counting Modulo Theories (Phan, 2015) leverages SMT solvers to compute model counts, but does not cover weighted model counting (and thus probabilistic reasoning), and does not exploit factorization. Weighted Model Counting and Integration (Michels *et al.*, 2015; Belle *et al.*, 2015; Michels *et al.*, 2016) and Symbolic Variable Elimination (Sanner and Abbasnejad, 2012) are similar in spirit to PIMT, but not to LPIMT; they apply to Boolean and linear real arithmetic random variables, but not yet to random functions. (Belle, 2017) focuses on weighted model counting (WMC) with function symbols on infinite domains, reducing it to standard WMC, but does not currently use lifting techniques. Group Inversion (Taghipour *et al.*, 2012) expands Inversion to cover some extra cases, but not interpreted functions. Extending it to do so, along with Counting, is the most immediate possibility for future work.

To conclude, we have defined Inversion Modulo Theories, an expansion of the lifted inference operation Inversion for the case with interpreted functions, and added it to Probabilistic Inference Modulo Theories framework, thus defining the first algorithm to perform exact lifted inference in the presence of interpreted functions.

References

Barrett, C. W., Sebastiani, R., Seshia, S. A., and Tinelli, C. Satisfiability Modulo Theories. In Biere, A., Heule, M., van Maaren, H., and Walsh, T., editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 825–885. IOS Press, 2009.

Belle, V., Passerini, A., and Van den Broeck, G. Probabilistic inference in hybrid domains by weighted model integration. In *Proceedings of 24th International Joint Conference on Artificial Intelligence (IJCAI)*, 2015.

Belle, V. Weighted model countingwith function symbols. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, 2017.

Bistarelli, S., Montanari, U., and Rossi, F. Semiring-based constraint satisfaction and optimization. *J. ACM*, 44(2):201–236, March 1997.

Boutilier, C., Friedman, N., Goldszmidt, M., and Koller, D. Context-Specific Independence in Bayesian Networks. In *Proceedings of UAI*, pages 115–123, 1996.

Carette, J. and Shan, C. Simplifying probabilistic programs using computer algebra. In *Practical Aspects of Declarative Languages - 18th International Symposium, PADL 2016, St. Petersburg, FL, USA, January 18-19, 2016. Proceedings*, pages 135–152, 2016.

de Moura, L., Dutertre, B., and Shankar, N. A tutorial on satisfiability modulo theories. In *Computer Aided Verification, 19th International Conference, CAV 2007, Berlin, Germany, July 3-7, 2007, Proceedings*, volume 4590 of *Lecture Notes in Computer Science*, pages 20–36. Springer, 2007.

de Salvo Braz, R., O'Reilly, C., Gogate, V., and Dechter, R. Probabilistic Inference Modulo Theories. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, New York, USA, 2016.

de Salvo Braz, R. *Lifted First-Order Probabilistic Inference*. PhD thesis, University of Illinois at Urbana-Champaign, 2007.

Goodman, N. D., Mansinghka, V. K., Roy, D. M., Bonawitz, K., and Tarlow, D. Church: a language for generative models. *CoRR*, abs/1206.3255, 2012.

Kersting, K. Lifted probabilistic inference. In *European Conference on Artificial Intelligence*, 2012.

Michels, S., Hommersom, A., Lucas, P. J. F., and Velikova, M. A new probabilistic constraint logic programming language based on a generalised distribution semantics. *Artificial Intelligence*, 228(C):1–44, November 2015.

Michels, S., Hommersom, A., and Lucas, P. J. F. Approximate probabilistic inference with bounded error. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, New York, USA, 2016.

Milch, B., Marthi, B., Russell, S., Sontag, D., Ong, D. L., and Kolobov, A. BLOG: probabilistic models with unknown objects. In *IJCAI'05: Proceedings of the 19th international joint conference on Artificial intelligence*, pages 1352–1359, San Francisco, CA, USA, 2005. Morgan Kaufmann Publishers Inc.

Milch, B., Zettlemoyer, L., Kersting, K., Haimes, M., and Kaelbling, L. P. Lifted probabilistic inference with counting formulas. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI-2008)*, Chicago, Illinois, USA, July 2008 2008.

Narayanan, P., Carette, J., Romano, W., Shan, C.-c., and Zinkov, R. Probabilistic inference by program transformation in hakaru (system description). In Kiselyov, O. and King, A., editors, *Functional and Logic Programming: 13th International Symposium, FLOPS 2016, Kochi, Japan, March 4-6, 2016, Proceedings*, pages 62–79. Springer International Publishing, Cham, 2016.

Pearl, J. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, San Mateo (Calif.), 1988.

Phan, Q.-S. *Model Counting Modulo Theories*. PhD thesis, Queen Mary University of London, 2015.

Poole, D. First-order probabilistic inference. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, pages 985–991, 2003.

Richardson, M. and Domingos, P. Markov Logic Networks. Technical report, Department of Computer Science, University of Washington, 2004.

Sanner, S. and Abbasnejad, E. Symbolic variable elimination for discrete and continuous graphical models. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012.

Taghipour, N., Fierens, D., den Broeck, G. V., Davis, J., and Blockeel, H. Lifted variable elimination: A novel operator and completeness results. *CoRR*, abs/1208.3809, 2012.

Van den Broeck, G., Taghipour, N., Meert, W., Davis, J., and Raedt, L. D. Lifted probabilistic inference by first-order knowledge compilation. In *In Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, pages 2178–2185, 2011.

Zhang, N. L. and Poole, D. A simple approach to Bayesian network computations. In *Proceedings of the Tenth Biennial Canadian Artificial Intelligence Conference*, 1994.