Inference in Probabilistic Logic Programs Using Lifted Explanations[*]

Arun Nampally[1] and C. R. Ramakrishnan[2]

1 **Computer Science Dept., Stony Brook University, Stony Brook, NY, USA**
`anampally@cs.stonybrook.edu`

2 **Computer Science Dept., Stony Brook University, Stony Brook, NY, USA**
`cram@cs.stonybrook.edu`

Abstract

In this paper, we consider the problem of lifted inference in the context of Prism-like probabilistic logic programming languages. Traditional inference in such languages involves the construction of an explanation graph for the query that treats each instance of a random variable separately. For many programs and queries, we observe that explanations can be summarized into substantially more compact structures introduced in this paper, called "lifted explanation graphs". In contrast to existing lifted inference techniques, our method for constructing lifted explanations naturally generalizes existing methods for constructing explanation graphs. To compute probability of query answers, we solve recurrences generated from the lifted graphs. We show examples where the use of our technique reduces the asymptotic complexity of inference.

1998 ACM Subject Classification D.1.6 Logic Programming, D.3.3 Language Constructs and Features – Constraints,F.3.2 Semantics of Programming Languages – Operational semantics, I.2.3 Deduction and Theorem Proving – Logic programming, Resolution, Uncertain, "fuzzy", and probabilistic reasoning

Keywords and phrases Probabilistic logic programs, Probabilistic inference, Lifted inference, Symbolic evaluation, Constraints

Digital Object Identifier 10.4230/OASIcs.ICLP.2016.15

1 Introduction

Probabilistic Logic Programming (PLP) provides a declarative programming framework to specify and use combinations of logical and statistical models. A number of programming languages and systems have been proposed and studied under the framework of PLP, e.g. PRISM [12], Problog [4], PITA [11] and Problog2 [5] etc. These languages have similar declarative semantics based on the *distribution semantics* [13]. The inference algorithms used in many of these systems to evaluate the probability of query answers, e.g. PRISM, Problog and PITA, are based on a common notion of *explanation graphs*. These graphs represent explanations, which are sets of *probabilistic choices* that are abduced during query evaluation. Explanation graphs are implemented differently by different systems; e.g. PRISM uses tables to represent them under *mutual exclusion* and *independence* assumptions on explanations; ProbLog and PITA represents them using Binary Decision Diagrams (BDDs).

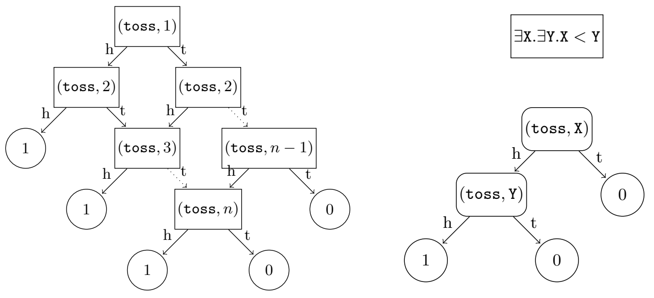
Inference based on explanation graphs does not scale well to logical/statistical models with large numbers of random processes and variables. In particular, in models containing *families*

[*] This work was supported in part by NSF grants IIS 1447549 and CNS 1405641.

```
% Two distinct tosses show "h"
twoheads :-
    X in coins,
    msw(toss, X, h),
    Y in coins,
    {X < Y},
    msw(toss, Y, h).

% Cardinality of coins:
:- population(coins, 100).

% Distribution parameters:
:- set_sw(toss,
categorical([h:0.5, t:0.5])).
```



(a) Simple Px program (b) Ground expl. Graph (c) Lifted expl. Graph

Figure 1 Example program and ground explanation graph.

of independent, identically distributed (i.i.d) random variables, outcomes of individual random variables are abduced. However, as developments in the area of *lifted inference* [10, 2, 7] have shown, vast savings in computational effort can be made by exploiting the symmetry in models with populations of i.i.d random variables. The lifted inference algorithms seek to treat a set of i.i.d random variables as single unit and aggregate their behavior to achieve computational speedup. *This paper presents a structure for representing explanation graphs compactly by exploiting the symmetry with respect to i.i.d random variables, and a procedure to build this structure without enumerating each instance of a random process.*

Illustration. The simple example in Fig. 1 shows a program describing a process of tossing a number of i.i.d. coins, and evaluating if at least two of them came up "heads". The example is specified in an extension of the PRISM language, called Px. Explicit random processes of PRISM enables a clearer exposition of our approach. In PRISM and Px, a special predicate of the form `msw(`p`, `i`, `v`)` describes, given a random process p that defines a family of i.i.d. random variables, that v is the value of the i-th random variable in the family. The argument i of `msw` is called the ith *instance* argument. In this paper, we consider Param-Px, a further extension of Px to define parameterized programs. In Param-Px, a built-in predicate, `in` is used to specify membership; e.g. x `in` s means x is member of an enumerable set s. The size of s is specified by a separate `population` directive. The program in Fig. 1 defines a family of random variables generated by `toss`. The instances that index these random variables are drawn from the set `coins`. Finally, predicate `twoheads` is defined to hold if tosses of at least two distinct coins come up "`h`".

State of the Art, and Our Solution. Inference in PRISM, Problog and PITA follows the structure of the derivations for a query. Consider the program in Fig. 1(a) and let the cardinality of the set of coins be n. The query `twoheads` will take $\Theta(n^2)$ time, since it will construct bindings to both `X` and `Y` in the clause defining `twoheads`. However, the size of an explanation graph is $\Theta(n)$, as shown in Fig. 1(b). Computing the probability of the query over this graph will also take $\Theta(n)$ time.

In this paper, we present a technique to construct a symbolic version of an explanation graph, called a *lifted explanation graph* that represents instances symbolically and avoids enumerating the instances of random processes such as `toss`. The lifted explanation graph for query `twoheads` is shown in Fig. 1(c). Unlike traditional explanation graphs where nodes are specific instances of random variables, nodes in the lifted explanation graph may be parameterized by their instance (e.g ($\texttt{toss}, X$) instead of ($\texttt{toss}, 1$)). A set of constraints on

$$
\begin{aligned}
f_1(n) &= h_1(1,n) \\
h_1(i,n) &= \begin{cases} g_1(i,n) + (1-\widehat{f}_1)\cdot h_1(i+1,n) & \text{if } i < n \\ g_1(i,n) & \text{if } i = n \end{cases} \\
g_1(i,n) &= \pi \cdot f_2(i,n) \\
\widehat{f}_1 &= \pi
\end{aligned}
\qquad
\begin{aligned}
f_2(i,n) &= \begin{cases} h_2(i+1,n) & \text{if } i < n \\ 0 & \text{otherwise} \end{cases} \\
h_2(j,n) &= \begin{cases} g_2 + (1-\widehat{f}_2)\cdot h_2(j+1,n) & \text{if } j < n \\ g_2 & \text{if } j = n \end{cases} \\
g_2 &= \pi \\
\widehat{f}_2 &= \pi
\end{aligned}
$$

Figure 2 Recurrences for computing probabilities for Example in Fig. 1.

those variables, specify the allowed groundings.

Note that the graph size is independent of the size of the population. Moreover, the graph can be constructed in time independent of the population size as well. Probability computation is performed by first deriving recurrences based on the graph's structure and then solving the recurrences. The recurrences for probability computation derived from the graph in Fig. 1(c) are shown in Fig. 2. In the figure, the equations with subscript 1 are derived from the root of the graph; those with subscript 2 from the left child of the root; and where π is the probability that `toss` is "`h`". Note that the probability of the query, $f_1(n)$, can be computed in $\Theta(n)$ time from the recurrences.

Contributions. The technical contribution of this paper is two fold.

1. We define a lifted explanation structure, and operations over these structures (see Section 3). We also give method to construct such structures during query evaluation, closely following the techniques used to construct explanation graphs.
2. We define a technique to compute probabilities over such structures by deriving and solving recurrences (see Section 4). We provide examples to illustrate the complexity gains due to our technique over traditional inference.

The rest of the paper begins by defining parameterized Px programs and their semantics (Section 2). After presenting the main technical work, the paper concludes with a discussion of related work. (Section 5).

2 Parameterized Px Programs

The PRISM language follows Prolog's syntax. It adds a binary predicate `msw` to introduce random variables into an otherwise familiar Prolog program. Specifically, in `msw(`s`, `v`)`, s is a "switch" that represents a random process which generates a family of random variables, and v is bound to the value of a variable in that family. The domain and distribution of the switches are specified by `set_sw` directives. Given a switch s, we use D_s to denote the domain of s, and $\pi_s : D_s \to [0,1]$ to denote its probability distribution.

2.1 Px and Inference

The Px language extends the PRISM language in three ways. Firstly, the `msw` switches in Px are ternary, with the addition of an explicit *instance* parameter. This brings the language closer to the formalism presented when describing PRISM's semantics [13]. Secondly, Px aims to compute the distribution semantics without the *mutual exclusion* and *independence* assumptions on explanations imposed by PRISM system. Thirdly, in contrast to PRISM, the switches in Px can be defined with a wide variety of univariate distributions, including continuous distributions (such as Gaussian) and infinite discrete distributions (such as Poisson). However, in this paper, we consider only programs with finite discrete distributions.

Exact inference of Px programs with finite discrete distributions uses explanation graphs with the following structure.

▶ **Definition 1** (Ground Explanation Graph). Let S be the set of ground switches in a Px program P, and D_s be the domain of switch $s \in S$. Let $\mathcal{T}$ be the set of all ground terms over symbols in P. Let "$\prec$" be a total order over $S \times \mathcal{T}$ such that $(s_1, t_1) \prec (s_2, t_2)$ if either $t_1 < t_2$ or $t_1 = t_2$ and $s_1 < s_2$. A *ground explanation tree* over P is a rooted tree γ such that:

- Leaves in γ are labeled 0 or 1.
- Internal nodes in γ are labeled (s, z) where $s \in S$ is a switch, and z is a ground term over symbols in P.
- For node labeled (s, z), there are k outgoing edges to subtrees, where $k = |D_s|$. Each edge is labeled with a unique $v \in D_s$.
- Let $(s_1, z_1), (s_2, z_2), \ldots, (s_k, z_k), c$ be the sequence of node labels in a root-to-leaf path in the tree, where $c \in \{0, 1\}$. Then $(s_i, z_i) \prec (s_j, z_j)$ if $i < j$ for all $i, j \leq k$. As a corollary, node labels along any root to leaf path in the tree are unique.

An *explanation graph* is a DAG representation of a ground explanation tree.

Consider a sequence of alternating node and edge labels in a root-to-leaf path: $(s_1, z_1), v_1, (s_2, z_2), v_2, \ldots, (s_k, z_k), v_k, c$. Each such path enumerates a set of random variable valuations $\{s_1[z_1] = v_1, s_2[z_2] = v_2, \ldots, s_k[z_k] = v_k\}$. When $c = 1$, the set of valuations forms an explanation. An explanation graph thus represents a set of explanations.

Note that explanation trees and graphs resemble decision diagrams. Indeed, explanation graphs are implemented using Binary Decision Diagrams [3] in PITA and Problog; and Multi-Valued Decision Diagrams [15] in Px. The *union* of two sets of explanations can be seen as an "*or*" operation over corresponding explanation graphs. Pair-wise union of explanations in two sets is an "*and*" operation over corresponding explanation graphs.

2.1.1 Inference via Program Transformation

Inference in Px is performed analogous to that in PITA [11]. Concretely, inference is done by translating a Px program to one that explicitly constructs explanation graphs, performing tabled evaluation of the derived program, and computing probability of answers from the explanation graphs. We describe the translation for definite pure programs; programs with built-ins and other constructs can be translated in a similar manner.

For every user-defined atom A of the form $p(t_1, t_2, \ldots, t_n)$, we define $\mathit{exp}(A, E)$ as atom $p(t_1, t_2, \ldots, t_n, E)$ with a new predicate $p/(n+1)$, with E as an added "explanation" argument. For such atoms A, we also define $\mathit{head}(A, E)$ as atom $p'(t_1, t_2, \ldots, t_n, E)$ with a new predicate $p'/(n+1)$. A *goal* G is a conjunction of atoms, where $G = (G_1, G_2)$ for goals G_1 and G_2, or G is an atom A. Function *exp* is extended to goals such that $\mathit{exp}((G_1, G_2)) = ((\mathit{exp}(G_1, E_1), \mathit{exp}(G_2, E_2)), \mathtt{and}(E_1, E_2, E))$, where `and` is a predicate in the translated program that combines two explanations using conjunction, and E_1 and E_2 are fresh variables. Function *exp* is also extended to `msw` atoms such that $\mathit{exp}(\mathtt{msw}(p, i, v), E)$ is $\mathtt{rv}(p, i, v, E)$, where `rv` is a predicate that binds E to an explanation graph with root labeled (p, i) with an edge labeled v leading to a 1 child, and all other edges leading to 0.

Each clause of the form A :– G in a Px program is translated to a new clause $\mathit{head}(A, E)$:– $\mathit{exp}(G, E)$. For each predicate p/n, we define $p(X_1, X_2, \ldots X_n, E)$ to be such that E is the disjunction of all E' for $p'(X_1, X_2, \ldots X_n, E')$. As in PITA, this is done using answer subsumption.

Probability of an answer is determined by first materializing the explanation graph, and then computing the probability over the graph. The probability associated with a node in the graph is computed as the sum of the products of probabilities associated with its children and the corresponding edge probabilities. The probability associated with an explanation graph φ, denoted $prob(\varphi)$ is the probability associated with the root. This can be computed in time linear in the size of the graph by using dynamic programming or tabling.

2.2 Syntax and Semantics of Parameterized Px Programs

Parameterized Px, called Param-Px for short, is a further extension of the Px language. The first feature of this extension is the specification of *populations* and *instances* to specify ranges of instance parameters of `msw`s.

▶ **Definition 2** (Population). A *population* is a named finite set, with a specified cardinality. A population has the following properties:

1. Elements of a population may be atomic, or depth-bounded ground terms.
2. Elements of a population are totally ordered using the default term order.
3. Distinct populations are disjoint.

Populations and their cardinalities are specified in a Param-Px program by `population` facts. For example, the program in Figure 1(a) defines a population named `coins` of size 100. The individual elements of this set are left unspecified. When necessary, `element/2` facts may be used to define distinguished elements of a population. For example, `element(fred, persons)` defines a distinguished element "`fred`" in population persons. In presence of `element` facts, elements of a population are ordered as follows. The order of `element` facts specifies the order among the distinguished elements, and all distinguished elements occur before other unspecified elements in the order.

▶ **Definition 3** (Instance). An *instance* is an element of a population. In a Param-Px program, a built-in predicate `in/2` can be used to draw an instance from a population. All instances of a population can be drawn by backtracking over `in`.

An *instance variable* is one that occurs as the instance argument in a `msw` predicate in a clause of a Param-Px program. In Fig. 1(a), `X in coins` binds `X` to an instance of population `coins` and `X`, `Y` are instance variables.

Constraints. The second extension in Param-Px are atomic constraints, of the form $\{t_1 = t_2\}$, $\{t_1 \neq t_2\}$ and $\{t_1 < t_2\}$, where t_1 and t_2 are variables or constants, to compare instances of a population. We use braces "$\{\cdot\}$" to distinguish the constraints from Prolog built-in comparison operators. In Figure 1(a), `{X \= Y}` is an atomic constraint.

Types. We use populations in a Param-Px program to confer types to program variables. Each variable that occurs in an "`in`" predicate is assigned a unique type. More specifically, X has type p if X `in` p occurs in a program, where p is a population; and X is untyped otherwise. We extend this notion of types to constants and switches as well. A constant c has type p if there is a fact `element(`c`,` p`)`; and c is untyped otherwise. A switch s has type p if there is an `msw(`s`,` X`,` t`)` in the program and X has type p; and s is untyped otherwise.

▶ **Definition 4** (Well-typedness). A Param-Px program is *well-typed* if:

1. For every constraint in the program of the form $\{t_1 = t_2\}$, $\{t_1 \neq t_2\}$ or $\{t_1 < t_2\}$, the types of t_1 and t_2 are identical.
2. Types of arguments of every atom on the r.h.s. of a clause are identical to the types of corresponding parameters of l.h.s. atoms of matching clauses.
3. Every switch in the program has a unique type.

The first two conditions of well-typedness ensure that only instances from the same population are compared in the program. The last condition imposes that instances of random variables generated by switch s are all indexed by elements drawn from the same population. In the rest of the paper, unless otherwise specified, we assume all Param-Px programs under consideration are well-typed.

Semantics of Param-Px Programs. Each Param-Px program can be readily transformed into a non-parameterized "ordinary" Px program. Each `population` fact is used to generate a set of `in/2` facts enumerating the elements of the population. Other constraints are replaced by their counterparts is Prolog: e.g. $\{X < Y\}$ with X<Y. Finally, each `msw(`s`,`i`,`t`)` is preceded by i `in` p where p is the type of s. The semantics of the original parameterized program is defined by the semantics of the transformed program.

3 Lifted Explanations

In this section we formally define *lifted explanation graphs*. These are a generalization of *ground explanation graphs* defined earlier, and are introduced in order to represent ground explanations compactly. Constraints over instances form a basic building block of lifted explanations and the following constraint language is used for the purpose.

3.1 Constraints on Instances

▶ **Definition 5** (Instance Constraints)**.** Let $\mathcal{V}$ be a set of instance variables, with subranges of integers as domains, such that m is the largest positive integer in the domain of any variable. Atomic constraints on instance variables are of one of the following two forms: $X < aY \pm k$, $X = aY \pm k$, where $X, Y \in \mathcal{V}$, $a \in 0, 1$, where k is a non-negative integer $\leq m + 1$. The language of constraints over bounded integer intervals, denoted by $\mathcal{L}(\mathcal{V}, m)$, is a set of formulae η, where η is a non-empty set of atomic constraints representing their conjunction.

Note that each formula in $\mathcal{L}(\mathcal{V}, m)$ is a convex region in $\mathbb{Z}^{|V|}$, and hence is closed under conjunction and existential quantification.

Let $vars(\eta)$ be the set of instance variables in an instance constraint η. A substitution $\sigma : vars(\eta) \to [1..m]$ that maps each variable to an element in its domain is a *solution* to η if each constraint in η is satisfied by the mapping. The set of all solutions of η is denoted by $[\![\eta]\!]$. The constraint formula η is unsatisfiable if $[\![\eta]\!] = \emptyset$. We say that $\eta \models \eta'$ if every $\sigma \in [\![\eta]\!]$ is a solution to η'.

Note also that instance constraints are a subclass of the well-known integer octagonal constraints [8] and can be represented canonically by difference bound matrices (DBMs) [18, 6], permitting efficient algorithms for conjunction and existential quantification. Given a constraint on n variables, a DBM is a $(n+1) \times (n+1)$ matrix with rows and columns indexed by variables (and a special "zero" row and column). For variables X and Y, the entry in cell (X, Y) of a DBM represents the upper bound on $X - Y$. For variable X, the value at cell $(X, 0)$ is X's upper bound and the value at cell $(0, X)$ is the negation of X's lower bound.

Geometrically, each entry in the DBM representing a η is a "face" of the region representing $[\![\eta]\!]$. Negation of an instance constraint η can be represented by a set of mutually exclusive instance constraints. Geometrically, this can be seen as the set of convex regions obtained by complementing the "faces" of the region representing $[\![\eta]\!]$. Note that when η has n variables, the number of instance constraints in $\neg\eta$ is bounded by the number of faces of $[\![\eta]\!]$, and hence by $O(n^2)$.

Let $\neg\eta$ represent the set of mutually exclusive instance constraints representing the negation of η. Then the disjunction of two instance constraints η and η' can be represented by the set of mutually exclusive instance constraints $(\eta \wedge \neg\eta') \cup (\eta' \wedge \neg\eta) \cup \{\eta \wedge \eta'\}$, where we overload $\wedge$ to represent the element-wise conjunction of an instance constraint with a set of constraints.

An existentially quantified formula of the form $\exists X.\eta$ can be represented by a DBM obtained by removing the rows and columns corresponding to X in the DBM representation of η. We denote this simple procedure to obtain $\exists X.\eta$ from η by $Q(X, \eta)$.

▶ **Definition 6** (Range)**.** Given a constraint formula $\eta \in \mathcal{L}(\mathcal{V}, m)$, and $X \in \mathit{vars}(\eta)$, let $\sigma_X(\eta) = \{v \mid \sigma \in [\![\eta]\!], \sigma(X) = v\}$. Then $\mathit{range}(X, \eta)$ is the interval $[l, u]$, where $l = \mathit{min}(\sigma_X(\eta))$ and $u = \mathit{max}(\sigma_X(\eta))$.

Since the constraint formulas represent convex regions, it follows that each variable's range will be an interval. Note that range of a variable can be readily obtained in constant time from the entries for that variable in the zero row and zero column of the constraint's DBM representation.

3.2 Lifted Explanation Graphs

▶ **Definition 7** (Lifted Explanation Graph)**.** Let S be the set of ground switches in a Param-Px program P, D_s be the domain of switch $s \in S$, m be the sum of the cardinalities of all populations in P and C be the set of distinguished elements of the populations in P. A *lifted explanation graph* over variables $\mathcal{V}$ is a pair $(\Omega : \eta, \psi)$ which satisfies the following conditions

1. $\Omega : \eta$ is the notation for $\exists\Omega.\eta$, where $\eta \in \mathcal{L}(\mathcal{V}, m)$ is either a satisfiable constraint formula, or the single atomic constraint `false` and $\Omega \subseteq \mathit{vars}(\eta)$ is the set of quantified variables in η. When η is `false`, $\Omega = \emptyset$.
2. ψ is a singly rooted DAG which satisfies the following conditions
   - Internal nodes are labeled (s, t) where $s \in S$ and $t \in \mathcal{V} \cup C$.
   - Leaves are labeled either 0 or 1.
   - Each internal node has an outgoing edge for each outcome $\in D_s$.
   - If a node labeled (s, t) has a child labeled (s', t') then $\eta \models t < t'$ or $\eta \models t = t'$ and $(s, c) \prec (s', c)$ for any ground term c (see Def. 1).

In this paper ground explanation graphs (Def. 1), and the DAG components of lifted explanation graphs are represented by textual patterns $(s, t)[\alpha_i : \psi_i]$ where (s, t) is the label of the root and ψ_i is the DAG associated with the edge labeled α_i. Irrelevant parts may denoted "_" to reduce clutter. We define the standard notion of bound and free variables over lifted explanation graphs.

▶ **Definition 8** (Bound and free variables)**.** Given a lifted explanation graph $(\Omega : \eta, \psi)$, a variable $X \in \mathit{vars}(\eta)$, is called a bound variable if $X \in \Omega$, otherwise its called a free variable.

The lifted explanation graph is said to be *well-structured* if every pair of nodes (s, X) and (s', X) with the same bound variable X, have a common ancestor with X as the instance variable. In the rest of the paper, we assume that the lifted explanation graphs are well-structured.

▶ **Definition 9** (Substitution operation). Given a lifted explanation graph $(\Omega : \eta, \psi)$, a variable $X \in \mathit{vars}(\eta)$, the substitution of X in the lifted explanation graph with a value k from its domain, denoted by $(\Omega : \eta, \psi)[k/X]$ is defined as follows: If $\eta[k/X]$ is unsatisfiable, then the result of the substitution is $(\emptyset : \{\texttt{false}\}, 0)$. If $\eta[k/X]$ is satisfiable, then $(\Omega : \eta, \psi)[k/X] = (\Omega \setminus \{X\} : \eta[k/X], \psi[k/X])$. The definition of $\psi[k/X]$ is as follows:

$$\begin{array}{ll|l} ((s,t)[\alpha_i : \psi_i])[k/X] = (s,k)[\alpha_i : \psi_i[k/X]], & \text{if } t = X & 0[k/X] = 0 \\ ((s,t)[\alpha_i : \psi_i])[k/X] = (s,t)[\alpha_i : \psi_i[k/X]], & \text{if } t \neq X & 1[k/X] = 1 \end{array}$$

The definition of substitution operation can be generalized to mappings on sets of variables in the obvious way.

▶ **Lemma 10** (Substitution lemma). *If $(\Omega : \eta, \psi)$ is a lifted explanation graph, and $X \in \mathit{vars}(\eta)$, then $(\Omega : \eta, \psi)[k/X]$ where k is a value in domain of X, is a lifted explanation graph.*

When a substitution $[k/X]$ is applied to a lifted explanation graph, and $\eta[k/X]$ is unsatisfiable, the result is $(\emptyset : \{\texttt{false}\}, 0)$ which is clearly a lifted explanation graph. When $\eta[k/X]$ is satisfiable, the variable is removed from Ω and occurrences of X in ψ are replaced by k. The resultant DAG clearly satisfies the conditions imposed by the Def. 7. Finally we note that a ground explanation graph ϕ (Def. 1) is a trivial lifted explanation graph $(\emptyset : \{\texttt{true}\}, \phi)$. This constitutes the informal proof of Lemma 10.

3.3 Semantics of Lifted Explanation Graphs

The meaning of a lifted explanation graph $(\Omega : \eta, \psi)$ is given by the ground explanation tree represented by it.

▶ **Definition 11** (Grounding). Let $(\Omega : \eta, \psi)$ be a closed lifted explanation graph, i.e., it has no free variables. Then the ground explanation tree represented by $(\Omega : \eta, \psi)$, denoted $\mathit{Gr}((\Omega : \eta, \psi))$, is given by the function $\mathit{Gr}(\Omega, \eta, \psi)$. When $[\![\eta]\!] = \emptyset$, then $\mathit{Gr}(\_, \eta, \_) = 0$. We consider the cases when $[\![\eta]\!] \neq \emptyset$. The grounding of leaves is defined as $\mathit{Gr}(\_, \_, 0) = 0$ and $\mathit{Gr}(\_, \_, 1) = 1$. When the instance argument of the root is a constant, grounding is defined as $\mathit{Gr}(\Omega, \eta, (s,t)[\alpha_i : \psi_i]) = (s,t)[\alpha_i : \mathit{Gr}(\Omega, \eta, \psi_i)]$. When the instance argument is a bound variable, the grounding is defined as $\mathit{Gr}(\Omega, \eta, (s,t)[\alpha_i : \psi_i]) \equiv \bigvee_{c \in range(t,\eta)} (s,c)[\alpha_i : \mathit{Gr}(\Omega \setminus \{t\}, \eta[c/t], \psi_i[c/t])]$.

In the above definition $\psi[c/t]$ represents the tree obtained by replacing every occurrence of t in the tree with c. The disjunct $(s,c)[\alpha_i : \mathit{Gr}(\Omega \setminus \{t\}, \eta[c/t], \psi_i[c/t])]$ in the above definition is denoted $\phi_{(s,c)}$ when the lifted explanation graph is clear from the context.

3.4 Operations on Lifted Explanation Graphs

And/Or Operations. Let $(\Omega : \eta, \psi)$ and $(\Omega' : \eta', \psi')$ be two lifted explanation graphs. We now define "$\wedge$" and "$\vee$" operations on them. The "$\wedge$" and "$\vee$" operations are carried out in two steps. First, the constraint formulas of the inputs are combined. However, the free variables in the operands may have *no known order* among them. Since, an arbitrary order cannot be imposed, the operations are defined in a *relational*, rather than functional form. We use the notation $(\Omega : \eta, \psi) \oplus (\Omega' : \eta', \psi') \rightarrow (\Omega'' : \eta'', \psi'')$ to denote that $(\Omega'' : \eta'', \psi'')$ is *a* result of $(\Omega : \eta, \psi) \oplus (\Omega' : \eta', \psi')$. When an operation returns multiple answers due to ambiguity on the order of free variables, the answers that are inconsistent with the final order are discarded. We assume that the variables in the two lifted explanation graphs are standardized apart such that the bound variables of $(\Omega : \eta, \psi)$ and $(\Omega' : \eta', \psi')$ are all distinct, and different from free variables of $(\Omega : \eta, \psi)$ and $(\Omega' : \eta', \psi')$. Let $\psi = (s,t)[\alpha_i : \psi_i]$ and $\psi' = (s',t')[\alpha'_i : \psi'_i]$.

Combining constraint formulae

$Q(\Omega, \eta) \wedge Q(\Omega', \eta')$ is unsatisfiable. Then the orders among free variables in η and η' are incompatible.

- The $\wedge$ operation is defined as $(\Omega : \eta, \psi) \wedge (\Omega' : \eta', \psi') \to (\emptyset : \{\texttt{false}\}, 0)$
- The $\vee$ operation simply returns the two inputs as outputs:

$$\begin{aligned}(\Omega : \eta, \psi) \vee (\Omega' : \eta', \psi') \to& (\Omega : \eta, \psi)\\ (\Omega : \eta, \psi) \vee (\Omega' : \eta', \psi') \to& (\Omega' : \eta', \psi')\end{aligned}$$

$Q(\Omega, \eta) \wedge Q(\Omega', \eta')$ is satisfiable. The orders among free variables in η and η' are compatible

- The $\wedge$ operation is defined as $(\Omega : \eta, \psi) \wedge (\Omega' : \eta', \psi') \to (\Omega \cup \Omega' : \eta \wedge \eta', \psi \wedge \psi')$.
- The $\vee$ operation is defined as

$$\begin{aligned}(\Omega : \eta, \psi) \vee (\Omega' : \eta', \psi') \to& (\Omega \cup \Omega' : \eta \wedge \neg\eta', \psi)\\ (\Omega : \eta, \psi) \vee (\Omega' : \eta', \psi') \to& (\Omega \cup \Omega' : \eta' \wedge \neg\eta, \psi')\\ (\Omega : \eta, \psi) \vee (\Omega' : \eta', \psi') \to& (\Omega \cup \Omega' : \eta \wedge \eta', \psi \vee \psi')\end{aligned}$$

Combining DAGs. Now we describe $\wedge$ and $\vee$ operations on the two DAGs ψ and ψ' in the presence of a single constraint formula. The general form of the operation is $(\Omega : \eta, \psi \oplus \psi')$.

Base cases: The base cases are as follows (symmetric cases are defined analogously).

$(\Omega : \eta, 0 \vee \psi') \to (\Omega : \eta, \psi')$	$(\Omega : \eta, 0 \wedge \psi') \to (\Omega : \eta, 0)$
$(\Omega : \eta, 1 \vee \psi') \to (\Omega : \eta, 1)$	$(\Omega : \eta, 1 \wedge \psi') \to (\Omega : \eta, \psi')$

Recursion: When the base cases do not apply, we try to compare the roots of ψ and ψ'. The root nodes are compared as follows: We say $(s, t) = (s', t')$ if $\eta \models t = t'$ and $s = s'$, else $(s, t) < (s', t')$ (analogously $(s', t') < (s, t)$) if $\eta \models t < t'$ or $\eta \models t = t'$ and $(s, c) \prec (s', c)$ for any ground term c. If neither of these two relations hold, then the roots are not comparable and its denoted as $(s, t) \not\sim (s', t')$.

a. **$(s, t) < (s', t')$:** $(\Omega : \eta, \psi \oplus \psi') \to (\Omega : \eta, (s, t)[\alpha_i : \psi_i \oplus \psi'])$

b. **$(s', t') < (s, t)$:** $(\Omega : \eta, \psi \oplus \psi') \to (\Omega : \eta, (s', t')[\alpha'_i : \psi \oplus \psi'_i])$

c. **$(s, t) = (s', t')$:** $(\Omega : \eta, \psi \oplus \psi') \to (\Omega : \eta, (s, t)[\alpha_i : \psi_i \oplus \psi'_i])$

d. **$(s, t) \not\sim (s', t')$:** Operations depend on whether t, t' are free, bound or constant.

i. t is a free variable or a constant, and t' is a free variable (the symmetric case is analogous).

$$\begin{aligned}(\Omega : \eta, \psi \oplus \psi') \to& (\Omega : \eta \wedge t < t', \psi \oplus \psi')\\ (\Omega : \eta, \psi \oplus \psi') \to& (\Omega : \eta \wedge t = t', \psi \oplus \psi')\\ (\Omega : \eta, \psi \oplus \psi') \to& (\Omega : \eta \wedge t' < t, \psi \oplus \psi')\end{aligned}$$

ii. t is a free variable or a constant and t' is a bound variable $(\Omega : \eta, \psi \oplus \psi')$ is defined as (the symmetric case is analogous):

$$(\Omega : \eta \wedge t < t', \psi \oplus \psi') \vee (\Omega : \eta \wedge t = t', \psi \oplus \psi') \vee (\Omega : \eta \wedge t' < t, \psi \oplus \psi')$$

Note that in the above definition, all three lifted explanation graphs use the same variable names for bound variable t'. Lifted explanation graphs can be easily standardized apart on the fly, and henceforth we assume that the operation is applied as and when required.

iii. t and t' are bound variables. Let $range(t, \eta) = [l_1, u_1]$ and $range(t', \eta) = [l_2, u_2]$. We can conclude that $range(t, \eta)$ and $range(t', \eta)$ are overlapping, otherwise (s, t) and (s', t') could have been ordered. Without loss of generality, we assume that $l_1 \leq l_2$. The various cases of overlap and the corresponding definition of the $(\Omega : \eta, \psi \oplus \psi')$ is given in the following table.

$l_1 = l_2, u_1 = u2$	$(\Omega \cup \{t''\} : \eta \wedge l_1 - 1 < t'' \wedge t'' - 1 < u_1 \wedge t'' < t \wedge t'' < t', (s, t'')[\alpha_i : (\psi_i[t''/t] \oplus \psi'_i[t''/t']) \vee (\psi_i[t''/t] \oplus \psi') \vee (\psi'_i[t''/t'] \oplus \psi)])$
$l_1 = l_2, u_1 < u_2$	$(\Omega : \eta \wedge t' - 1 < u_1, \psi \oplus \psi') \vee (\Omega : \eta \wedge u_1 < t', \psi \oplus \psi')$
$l_1 = l_2, u_2 < u_1$	$(\Omega : \eta \wedge t = t', \psi \oplus \psi') \vee (\Omega : \eta \wedge u_2 < t, \psi \oplus \psi')$
$l_1 < l_2, u_1 = u_2$	$(\Omega : \eta \wedge t = t', \psi \oplus \psi') \vee (\Omega : \eta \wedge t < l_2, \psi \oplus \psi')$
$l_1 < l_2, u_1 < u_2$	$(\Omega : \eta \wedge u_1 < t', \psi \oplus \psi') \vee (\Omega : \eta \wedge t < l_2 \wedge t' - 1 < u_1, \psi \oplus \psi') \vee (\Omega : \eta \wedge t = t', \psi \oplus \psi')$
$l_1 < l_2, u_2 < u_1$	$(\Omega : \eta \wedge u_2 < t, \psi \oplus \psi') \vee (\Omega : \eta \wedge t < l_2, \psi \oplus \psi') \vee (\Omega : \eta \wedge t = t', \psi \oplus \psi')$

▶ **Lemma 12** (Correctness of "$\wedge$" and "$\vee$" operations)**.** *Let $(\Omega : \eta, \psi)$ and $(\Omega' : \eta', \psi')$ be two lifted explanation graphs with free variables $\{X_1, X_2 \ldots, X_n\}$. Let Σ be the set of all substitutions mapping each X_i to a value in its domain. Then, for every $\sigma \in \Sigma$ and $\oplus \in \{\wedge, \vee\}$, $Gr(((\Omega : \eta, \psi) \oplus (\Omega' : \eta', \psi'))\sigma) = Gr((\Omega : \eta, \psi)\sigma) \oplus Gr((\Omega' : \eta', \psi')\sigma)$*

Quantification

▶ **Definition 13** (Quantification)**.** Let $(\Omega : \eta, \psi)$ be a lifted inference graph and $X \in vars(\eta)$. Then $quantify((\Omega : \eta, \psi), X) = (\Omega \cup \{X\} : \eta, \psi)$.

▶ **Lemma 14** (Correctness of *quantify*)**.** *Let $(\Omega : \eta, \psi)$ be a lifted explanation graph, let σ_{-X} be a substitution mapping all the free variables in $(\Omega : \eta, \psi)$ except X to values in their domains. Let Σ be the set of mappings σ such that σ maps all free variables to values in their domains and is identical to σ_{-X} at all variables except X. Then the following holds $Gr(quantify((\Omega : \eta, \psi), X)\sigma_{-X}) = \bigvee_{\sigma \in \Sigma} Gr((\Omega : \eta, \psi)\sigma)$*

Construction of Lifted Explanation Graphs. Lifted explanation graphs for a query are constructed by transforming the Param-Px program $\mathcal{P}$ into one that explicitly constructs a lifted explanation graph, following a similar procedure to the one outlined in Section 2 for constructing ground explanation graphs. The main difference is the use of existential quantification. Let $A :- G$ be a program clause, and $vars(G) - vars(A)$ be the set of variables in G and not in A. If any of these variables has a type, then it means that the variable used as an instance argument in G is existentially quantified. Such clauses are then translated as $head(A, E_h) :- exp(G, E_g), quantify(E_g, V_s, E_h)$, where V_s is the set of typed variables in $vars(G) - vars(A)$. A minor difference is the treatment of constraints: *exp* is extended to atomic constraints φ such that $exp(\varphi, E)$ binds E to $(\emptyset : \{\varphi\}, 1)$.

We order the populations and map the elements of the populations to natural numbers as follows. The population that comes first in the order is mapped to natural numbers in the range $1..m$, where m is the cardinality of this population. Any constants in this population are mapped to natural numbers in the low end of the range. The next population in the order is mapped to natural numbers starting from $m + 1$ and so on. Thus, each typed variable is assigned a domain of contiguous positive values. The rest of the program transformation remains the same, the underlying graphs are constructed using the lifted operators. The lifted explanation graph corresponding to the query in Fig 1(a) is shown in Fig 1(c).

4 Lifted Inference using Lifted Explanations

In this section we describe a technique to compute answer probabilities in a lifted fashion from closed lifted explanation graphs. This technique works on a restricted class of lifted explanation graphs satisfying a property we call the *frontier subsumption property*.

▶ **Definition 15** (Frontier)**.** Given a closed lifted explanation graph $(\Omega : \eta, \psi)$, the frontier of ψ w.r.t $X \in \Omega$ denoted $\mathit{frontier}_X(\psi)$ is the set of non-zero maximal subtrees of ψ, which do not contain a node with X as the instance variable.

Analogous to the set representation of explanations described in Section 2.1, we consider the set representations of lifted explanations, i.e., root-to-leaf paths in the DAGs of lifted explanation graphs that end in a "1" leaf. We consider *term substitutions* that can be applied to lifted explanations. These substitutions replace a variable by a term and further apply standard re-writing rules such as simplification of algebraic expressions. As before, we allow *term mappings* that specify a set of *term substitutions*.

▶ **Definition 16** (Frontier subsumption property)**.** A closed lifted explanation graph $(\Omega : \eta, \psi)$ satisfies the frontier subsumption property w.r.t $X \in \Omega$, if under term mappings $\sigma_1 = \{X \pm k + 1/Y \mid \langle X \pm k < Y \rangle \in \eta\}$ and $\sigma_2 = \{X + 1/X\}$, every tree $\phi \in \mathit{frontier}_X(\psi)$ satisfies the following condition: for every lifted explanation E_2 in ψ, there is a lifted explanation E_1 in ϕ such that $E_1\sigma_1$ is a sub-explanation (i.e., subset) of $E_2\sigma_2$.

A lifted explanation graph is said to satisfy frontier subsumption property, if it is satisfied for each bound variable. This property can be checked in a bottom up fashion for all bound variables in the graph. The tree obtained by replacing all subtrees in $\mathit{frontier}_X(\psi)$ by 1 in ψ is denoted $\widehat{\psi}_X$.

For closed lifted explanation graphs satisfying the above property, the probability of query answers can be computed using the following set of recurrences. With each subtree $\psi = (s, t)[\alpha_i : \psi_i]$ of the DAG of the lifted explanation graph, we associate function $f(\sigma, \psi)$ where σ is a (possibly incomplete) mapping of variables in Ω to values in their domains.

▶ **Definition 17** (Probability recurrences)**.** Given a closed lifted explanation graph $(\Omega : \eta, \psi)$, we define $f(\sigma, \psi)$ (as well as $g(\sigma, \psi)$ and $h(\sigma, \psi)$ wherever applicable) for a partial mapping σ of variables in Ω to values in their domains based on the structure of ψ. As before $\psi = (s, t)[\alpha_i : \psi_i]$

Case 1: ψ is a 0 leaf node. Then $f(\sigma, 0) = 0$

Case 2: ψ is a 1 leaf node. Then $f(\sigma, 1) = \begin{cases} 1, \text{ if } [\![\eta\sigma]\!] \neq \emptyset \\ 0, \text{ otherwise} \end{cases}$

Case 3: $t\sigma$ is a constant. Then $f(\sigma, \psi) = \begin{cases} \sum_{\alpha_i \in D_s} \pi_s(\alpha_i) \cdot f(\sigma, \psi_i), \text{ if } [\![\eta\sigma]\!] \neq \emptyset \\ 0, \text{ otherwise} \end{cases}$

Case 4: $t\sigma \in \Omega$, and $\mathit{range}(t, \eta\sigma) = (l, u)$. Then

$$f(\sigma, \psi) = \begin{cases} h(\sigma[l/t], \psi), \text{ if } [\![\eta\sigma]\!] \neq \emptyset \\ 0, \text{ otherwise} \end{cases}$$

$$h(\sigma[c/t], \psi) = \begin{cases} g(\sigma[c/t], \psi) + ((1 - P(\widehat{\psi}_t)) \times h(\sigma[c+1/t], \psi)), \text{ if } c < u \\ g(\sigma[c/t], \psi), \text{ if } c = u \end{cases}$$

$$g(\sigma, \psi) = \begin{cases} \sum_{\alpha_i \in D_s} \pi_s(\alpha_i) \cdot f(\sigma, \psi_i), \text{ if } [\![\eta\sigma]\!] \neq \emptyset \\ 0, \text{ otherwise} \end{cases}$$

In the above definition $\sigma[c/t]$ refers to a new partial mapping obtained by augmenting σ with the substitution $[c/t]$, $P(\widehat{\psi_t})$ is the sum of the probabilities of all branches leading to a 1 leaf in $\widehat{\psi_t}$. The functions f, g and h defined above can be readily specialized for each ψ. Moreover, the parameter σ can be replaced by the tuple of values actually used by a function. These rewriting steps yield recurrences such as those shown in Fig. 2. Note that $P(\widehat{\psi_t})$ can be computed using recurrences as well (shown as $\widehat{f}$ in Fig. 2).

▶ **Definition 18** (Probability of Lifted Explanation Graph)**.** Let $(\Omega : \eta, \psi)$ be a closed lifted explanation graph. Then, the probability of explanations represented by the graph, $prob((\Omega : \eta, \psi))$, is the value of $f(\{\}, \psi)$.

▶ **Theorem 19** (Correctness of Lifted Inference)**.** *Let $(\Omega : \eta, \psi)$ be a closed lifted explanation graph, and $\phi = Gr(\Omega : \eta, \psi)$ be the corresponding ground explanation graph. Then $prob((\Omega : \eta, \psi)) = prob(\phi)$.*

Given a closed lifted explanation graph, let k be the maximum number of instance variables along any root to leaf path. Then the function $f(\sigma, \psi)$ for the leaf will have to be computed for each mapping of the k variables. Each recurrence equation itself is either of constant size or bounded by the number of children of a node. Using dynamic programming a solution to the recurrence equations can be computed in polynomial time.

▶ **Theorem 20** (Efficiency of Lifted Inference)**.** *Let ψ be a closed lifted inference graph, n be the size of the largest population, and k be the largest number of instance variables along any root of leaf path in ψ. Then, $f(\{\}, \psi)$ can be computed in $O(|\psi| \times n^k)$ time.*

There are two sources of further optimization in the generation and evaluation of recurrences. First, certain recurrences may be transformed into closed form formulae which can be more efficiently evaluated. For instance, the closed form formula for $h(\sigma, \psi)$ for the subtree rooted at the node $(toss, Y)$ in Fig 1(c) can be evaluated in $O(\log(n))$ time while a naive evaluation of the recurrence takes $O(n)$ time. Second, certain functions $f(\sigma, \psi)$ need not be evaluated for every mapping σ because they may be independent of certain variables. For example, leaves are always independent of the mapping σ.

Other Examples. There are a number of simple probabilistic models that cannot be tackled by other lifted inference techniques but can be encoded in Param-Px and solved using our technique. For one such example, consider an urn with n balls, where the color of each ball is given by a distribution. Determining the probability that there are at least two green balls is easy to phrase as a directed first-order graphical model. However, lifted inference over such models can no longer be applied if we need to determine the probability of at least two green or two red balls. The probability computation for one of these events can be viewed as a generalization of noisy-OR probability computation, however dealing with the union requires the handling of intersection of the two events, due to which the $O(\log(N))$ time computation is no longer feasible.

For a more complex example, consider a system of n agents where each agent moves between various states in a stochastic manner. Consider a query to evaluate whether there are at least k agents in a given state s at a given time t. While this model is similar to a *collective graphical model* the aggregate query is different from those considered in [14], where computing probability of observed aggregate counts, parametering learning of individual model, and multiple path reconstruction are considered. Note that we cannot compile a model of this system into a clausal form without knowing the query. This system can be represented as a PRISM/Px program by modeling each agent's evolution as an independent

Hidden Markov Model (HMM). The lifted inference graph for querying the state of an arbitrary agent at time t is of size $O(e \cdot t)$, where e is the size of the transition relation of the HMM. For the "at least k agents" query, note that nodes in the lifted inference graph will be grouped by instances first, and hence the size of the graph (and the number of terms in the recurrences) is $O(k \cdot e \cdot t)$. The time complexity of evaluating the recurrences is $O(n \cdot k \cdot e \cdot t)$ where n is the total number of agents.

5 Related Work and Discussion

First-order graphical models [10, 2] are compact representations of propositional graphical models over populations. The key concepts in this field are that of *parameterized random variables* and *parfactors*. A parameterized random variable stands for a population of i.i.d. propositional random variables (obtained by grounding the logical variables). Parfactors are factors (potential functions) on parameterized random variables. By allowing large number of identical factors to be specified in a first-order fashion, first-order graphical models provide a representation that is independent of the population size. A key problem, then, is to perform *lifted* probabilistic inference over these models, i.e. without grounding the factors unnecessarily. The earliest such technique was *inversion elimination* presented in [10]. When summing out a parameterized random variable (i.e., all its groundings), it is observed that if all the logical variables in a parfactor are contained in the parameterized random variable, it can be summed out without grounding the parfactor.

The idea of *inversion elimination*, though powerful, exploits one of the many forms of symmetry present in first-order graphical models. Another kind of symmetry present in such models is that the values of an intermediate factor may depend on the histogram of propositional random variable outcomes, rather than their exact assignment. This symmetry is exploited by *counting elimination* [2] and elimination by *counting formulas* [7].

In [17] a form of lifted inference that uses constrained CNF theories with positive and negative weight functions over predicates as input was presented. Here the task of probabilistic inference in transformed to one of weighted model counting. To do the latter, the CNF theory is compiled into a structure known as first-order deterministic decomposable negation normal form. The compiled representation allows lifted inference by avoiding grounding of the input theory. This technique is applicable so long as the model can be formulated as a constrained CNF theory.

Another approach to lifted inference for probabilistic logic programs was presented in [1]. The idea is to convert a ProbLog program to parfactor representation and use a modified version of generalized counting first order variable elimination algorithm [16] to perform lifted inference. Problems where the model size is dependent on the query, such as models with temporal aspects, are difficult to solve with the knowledge compilation approach.

In this paper, we presented a technique for lifted inference in probabilistic logic programs using lifted explanation graphs. This technique is a natural generalization of inference techniques based on ground explanation graphs, and follows the two step approach: generation of an explanation graph, and a subsequent traversal to compute probabilities. A more complete description of this technique is in [9]. While the size of the lifted explanation graph is often independent of population, computation of probabilities may take time that is polynomial in the size of the population. A more sophisticated approach to computing probabilities from lifted explanation graph, by generating closed form formulae where possible, will enable efficient inference. Another direction of research would be to generate hints for lifted inference based on program constructs such as aggregation operators. Finally, our future work is focused on performing lifted inference over probabilistic logic programs that represent undirected and discriminative models.

Acknowledgments. We thank Andrey Gorlin for discussions and review of this work.

References

1 Elena Bellodi, Evelina Lamma, Fabrizio Riguzzi, Vítor Santos Costa, and Riccardo Zese. Lifted variable elimination for probabilistic logic programming. *TPLP*, 14(4-5):681–695, 2014.

2 Rodrigo De Salvo Braz, Eyal Amir, and Dan Roth. Lifted first-order probabilistic inference. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, pages 1319–1325, 2005.

3 Randal E Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys (CSUR)*, 24(3):293–318, 1992.

4 Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. ProbLog: A probabilistic prolog and its application in link discovery. In *IJCAI*, pages 2462–2467, 2007.

5 Anton Dries, Angelika Kimmig, Wannes Meert, Joris Renkens, Guy Van den Broeck, Jonas Vlasselaer, and Luc De Raedt. Problog2: Probabilistic logic programming. In *ECML PKDD*, pages 312–315, 2015.

6 Kim G. Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. Efficient verification of real-time systems: Compact data structure and state-space reduction. In *IEEE RTSS'97*, pages 14–24, 1997.

7 Brian Milch, Luke S Zettlemoyer, Kristian Kersting, Michael Haimes, and Leslie Pack Kaelbling. Lifted probabilistic inference with counting formulas. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, pages 1062–1068, 2008.

8 Antoine Miné. The octagon abstract domain. *Higher-Order and Symbolic Computation*, 19(1):31–100, 2006.

9 Arun Nampally and C. R. Ramakrishnan. Inference in probabilistic logic programs using lifted explanations. Technical report, Computer Science Department, Stony Brook University, 2016. URL: `http://www.cs.stonybrook.edu/~px/Papers/NR:lifted_tr_2016/`.

10 David Poole. First-order probabilistic inference. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, volume 3, pages 985–991, 2003.

11 Fabrizio Riguzzi and Terrance Swift. The PITA system: Tabling and answer subsumption for reasoning under uncertainty. *TPLP*, 11(4-5):433–449, 2011.

12 Taisuke Sato and Yoshitaka Kameya. PRISM: a language for symbolic-statistical modeling. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, volume 97, pages 1330–1339, 1997.

13 Taisuke Sato and Yoshitaka Kameya. Parameter learning of logic programs for symbolic-statistical modeling. *Journal of Artificial Intelligence Research*, pages 391–454, 2001.

14 Daniel R Sheldon and Thomas G. Dietterich. Collective graphical models. In *Advances in Neural Information Processing Systems 24*, pages 1161–1169, 2011. URL: `http://papers.nips.cc/paper/4220-collective-graphical-models.pdf`.

15 Arvind Srinivasan, Timothy Kam, Sharad Malik, and Robert K. Brayton. Algorithms for discrete function manipulation. In *International Conference on Computer-Aided Design, ICCAD*, pages 92–95, 1990. `doi:10.1109/ICCAD.1990.129849`.

16 Nima Taghipour, Daan Fierens, Jesse Davis, and Hendrik Blockeel. Lifted variable elimination: Decoupling the operators from the constraint language. *J. Artif. Intell. Res. (JAIR)*, 47:393–439, 2013.

17 Guy Van den Broeck, Nima Taghipour, Wannes Meert, Jesse Davis, and Luc De Raedt. Lifted probabilistic inference by first-order knowledge compilation. In *IJCAI*, pages 2178–2185, 2011.

18 Sergio Yovine. Model-checking timed automata. In *Embedded Systems*, number 1494 in LNCS, pages 114–152, 1998.