Inference in Probabilistic Logic Programs using Lifted Explanations∗

Arun Nampally and C. R. Ramakrishnan
Computer Science Department,
Stony Brook University, Stony Brook, NY 11794.
(*e-mail:* anampally@cs.stonybrook.edu, cram@cs.stonybrook.edu)

August 9, 2016

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 and computing probabilities over this graph. When evaluating queries over probabilistic logic programs with a large number of instances of random variables, traditional methods treat each instance separately. For many programs and queries, we observe that explanations can be summarized into substantially more compact structures, which we call lifted explanation graphs. In this paper, we define lifted explanation graphs and operations over them. 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.

1 Introduction

Background. 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 (Sato and Kameya 1997), Problog (De Raedt et al. 2007), PITA (Riguzzi and Swift 2011) and Problog2 (Dries et al. 2015) etc. These languages have similar declarative semantics based on the *distribution semantics* (Sato and Kameya 2001). Moreover, 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*.

At a high level, the inference procedure follows traditional query evaluation over logic programs. Outcomes of random variables, i.e., the *probabilistic choices*, are abduced during query

∗This work was supported in part by NSF Grant IIS-1447549.

evaluation. Each derivation of an answer is associated with a set of outcomes of random variables, called its explanation, under which the answer is supported by the derivation. Systems differ on how the explanations are represented and manipulated. Explanation graphs in PRISM are represented using tables, and under mutual exclusion assumption, multiple explanations are combined by adding entries to tables. In Problog and PITA, explanation graphs are represented by Binary Decision Diagrams (BDDs), with probabilistic choices mapped to propositional variables in BDDs.

Driving Problem. Inference based on explanation graphs does not scale well to logical/statistical models with large numbers of random processes and variables. Several *approximate inference* techniques have been proposed to estimate the probability of answers when exact inference is infeasible. In general, large logical/statistical models involve *families* of independent, identically distributed (i.i.d.) random variables. Moreover, in many models, inference often depends on the outcomes of random processes but not on the identities of random variables with the particular outcomes. However, query-based inference methods will instantiate each random variable and the explanation graph will represent each of their outcomes. Even when the graph may ultimately exhibit symmetry with respect to random variable identities, and many parts of the graph may be shared, the computation that produced these graphs may not be shared. *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. We illustrate the problem and our approach using the simple example in Figure 1, which 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 *instance* argument of the predicate. 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 Figure 1 defines a family of random variables with outcomes in {`h`, `t`} 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 "heads".

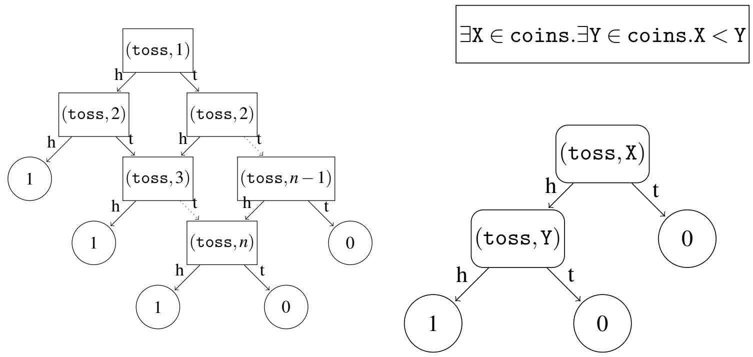
State of the Art, and Our Solution. Inference in PRISM, Problog and PITA follows the structure of the derivations for a query. Consider the program in Figure 1(a) and let the cardinality of the set of coins be n. The query `twoheads` will take $\Theta(n^2)$ time, since it will construct bindings to both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

```
% 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 Param-Px Program (b) Ground Explanation Graph (c) Lifted Explanation Graph

Figure 1: Example program and explanation graphs

the instances of random processes such as `toss`. The lifted explanation graph for query `twoheads` is shown in Figure 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 $(\mathtt{toss}, X)$ instead of $(\mathtt{toss}, 1)$). A set of constraints on 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 following recurrences capture the probability computation of the graph in Figure 1(c), where π is the probability that `toss` is h.

$$
\begin{aligned}
f(\{\}, \psi_1) &= h(\{1/X\}, \psi_1) && (1)\\
h(\{c/X\}, \psi_1) &= \begin{cases} g(\{c/X\}, \psi_1) + (1 - P(\widehat{\psi}_{1X})) \cdot h(\{c+1/X\}, \psi_1), & \text{if } c < u \\ g(\{c/X\}, \psi_1), & \text{if } c = u \end{cases} && (2)\\
g(\{c/X\}, \psi_1) &= \pi \cdot f(\{c/X\}, \psi_2) && (3)\\
P(\widehat{\psi}_{1X}) &= \pi && (4)\\
f(\{c/X\}, \psi_2) &= \begin{cases} h(\{c/X, c+1/Y\}, \psi_2), & \text{if } \eta\{c/X\} \text{ is satisfiable} \\ 0, & \text{otherwise} \end{cases} && (5)\\
h(\{c/X, d/Y\}, \psi_2) &= \begin{cases} g(\{c/X, d/Y\}, \psi_2) + (1 - P(\widehat{\psi}_{2Y})) \cdot h(\{c/X, d+1/Y\}, \psi_2), & \text{if } d < u \\ g(\{c/X, d/Y\}, \psi_2), & \text{if } d = u \end{cases} && (6)\\
g(\{c/X, d/Y\}, \psi_2) &= \pi && (7)\\
P(\widehat{\psi}_{2Y}) &= \pi && (8)
\end{aligned}
$$

These recurrences can be solved in $O(n)$ time with tabling or dynamic programming. Moreover, in certain cases, it is possible to obtain a closed form from a recurrence. For instance, noting that $g(\{c/X, d/Y\}, \psi_2)$ is independent of its parameters, we get $h(\{c/X, d/Y\}, \psi_2) = 1 - (1-\pi)^{n-c+1}$.

Lifted explanations vs. Lifted Inference. Our work is a form of *lifted inference*, a set of techniques that have been intensely studied in the context of first-order graphical models and Markov Logic Networks (Poole 2003; Braz et al. 2005; Milch et al. 2008). Essentially, lifted explanations provide a way to perform lifted inference over PLPs by leveraging their query evaluation mechanism. Directed first-order graphical models (Kisynski 2010) can be readily cast as PLPs, and our technique can be used to perform lifted inference over such models. Our solution, however, does not cover techniques based on counting elimination (Braz et al. 2005; Milch et al. 2008).

It should be noted that Problog2 does not construct query-specific explanation graphs. Instead, it uses a knowledge compilation approach where the models of a program are represented by a propositional boolean formula. These formulae, in turn, are represented in a compact standard form such as dDNNFs or SDDs (Darwiche 2001; Darwiche 2011). Query answers and their probabilities are then computed using linear-time algorithms over these structures.

The knowledge compilation approach has been extended to do a generalized form of lifted inference using first-order model counting (Van den Broeck et al. 2011). This technique performs lifted inference, including inversion and counting elimination over a large class of first order models. However, first order model counting is defined only when the problem can be stated in a first-order constrained CNF form. Problems such as the example in Figure 1 cannot be written in that form. To address this, a skolemization procedure which eliminates existential quantifiers and converts to first-order CNF without adding function symbols was proposed by Van den Broeck et al. (2014). While the knowledge compilation approach takes a core lifted inference procedure and moves to apply it to a class of logic programs, our approach generalizes existing inference techniques to perform a form of lifted inference.

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 parameters of the switches are specified using `value` facts and `set_sw` directives, respectively. 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.

The model-theoretic distribution semantics explicitly identifies each member of a random variable family with an *instance* parameter. In the PRISM system, the binary `msw` is interpreted stochastically, generating a new member of the random variable family whenever an `msw` is encountered during inference.

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 (Sato and Kameya 2001). Secondly, Px aims to compute the distribution semantics with no assumptions on the structure of the explanations. 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 $\mathscr{T}$ be the set of all ground terms over symbols in P. Let "$\prec$" be a total order over $S \times \mathscr{T}$ 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.* □

We use ϕ to denote explanation graphs. We use $(s,t)[v_i : \phi_i]$ to denote an explanation graph whose root is labeled (s,t), with each edge labeled v_i (ranging over a suitable index set i), leading to subgraph ϕ_i.

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 (Bryant 1992) in PITA and Problog; and Multi-Valued Decision Diagrams (Srinivasan et al. 1990) 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.

Inference via Program Transformation. Inference in Px is performed analogous to that in PITA (Riguzzi and Swift 2011). 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.

First every clause containing a disequality constraint is replaced by two clauses using less-than constraints. Next, for every user-defined atom A of the form $p(t_1,t_2,\ldots,t_n)$, we define $exp(A,E)$ 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 $head(A,E)$ 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 $exp((G_1,G_2)) = ((exp(G_1,E_1),exp(G_2,E_2)),\texttt{and}(E_1,E_2,E))$, 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 $exp(\texttt{msw}(p,i,v),E)$ is $\texttt{rv}(p,i,v,E)$, 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 $head(A,E)$:− $exp(G,E)$. 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.

Computing Answer Probabilities. Probability of an answer is determined by first materializing the explanation graph, and then computing the probability over the graph. The probability associated with an explanation graph φ is denoted by $prob(\varphi)$. This can be computed in time linear in the size of the graph by using dynamic programming or tabling to reuse computation results from shared subgraphs.

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`. □

For example, in Figure 1(a), `X in coins` binds `X` to an instance of population `coins`.

An *instance variable* is one that occurs as the instance argument in a `msw` predicate in a clause of a Param-Px program. For example, in Figure 1(a), `X` and `Y` in the clause defining `twoheads` 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.

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 and Typability)**.** *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.*

A Param-Px program is typable *if we can add literals of the form X `in` p (where p is a population) to r.h.s. of clauses such that the resulting program is well-typed.* □

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. As illustrated in Figure 1 in Introduction, the compactness of lifted explanations is a result of summarizing the instance information. Constraints over instances form a basic building block of lifted explanations. We use the following constraint domain for this purpose.

3.1 Constraints on Instances

Definition 5 (Instance Constraints)**.** *Let $\mathscr{V}$ 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 $X,Y \in \mathscr{V}$, $a \in 0,1$, where k is a non-negative integer $\leq m+1$. The language of constraints over bounded integer intervals, denoted by $\mathscr{L}(\mathscr{V},m)$, is a set of formulae η, where η is a non-empty set of atomic constraints representing their conjunction.*

Note that each formula in $\mathscr{L}(\mathscr{V},m)$ 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 $\sigma : vars(\eta) \rightarrow [1..m]$ 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 (Miné 2006) and can be represented canonically by difference bound matrices (DBMs) (Yovine 1998; LLP 1997), 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 $\eta \in \mathscr{L}(\mathscr{V},m)$, and $X \in vars(\eta)$, let $\sigma_X(\eta) = \{v \mid \sigma \in [\![\eta]\!], \sigma(X) = v\}$. Then range$(X,\eta)$ is the interval $[l,u]$, where $l = min(\sigma_X(\eta))$ and $u = max(\sigma_X(\eta))$.*

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 $\mathscr{V}$ is a pair $(\Omega : \eta, \psi)$ which satisfies the following conditions*

1. *$\Omega : \eta$ is the notation for $\exists\Omega.\eta$, where $\eta \in \mathscr{L}(\mathscr{V},m)$ is either a satisfiable constraint formula, or the single atomic constraint `false` and $\Omega \subseteq vars(\eta)$ 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 $t \in \mathscr{V} \cup C$.*
   - *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).*

Similar to ground explanation graphs (Def. 1), the DAG components of the 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 $X \in vars(\eta)$, 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* $X \in vars(\eta)$*, 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:*

$$
\begin{aligned}
(\Omega : \eta, \psi)[k/X] &= (\emptyset : \{\mathtt{false}\}, 0), \text{ if } \eta[k/X] \text{ is unsatisfiable} \\
(\Omega : \eta, \psi)[k/X] &= (\Omega \setminus \{X\} : \eta[k/X], \psi[k/X]), \text{ if } \eta[k/X] \text{ is satisfiable} \\
((s,t)[\alpha_i : \psi_i])[k/X] &= (s,k)[\alpha_i : \psi_i[k/X]], \text{ if } t = X \\
((s,t)[\alpha_i : \psi_i])[k/X] &= (s,t)[\alpha_i : \psi_i[k/X]], \text{ if } t \neq X \\
0[k/X] &= 0 \\
1[k/X] &= 1
\end{aligned}
$$

In the above definition, $\eta[k/X]$ refers to the standard notion of substitution. The definition of substitution operation can be generalized to mappings on sets of variables. Let σ be a substitution that maps variables to their values. By $(\Omega : \eta, \psi)\sigma$ we denote the lifted explanation graph obtained by sequentially performing substitution operation on each variable X in the domain of σ.

Lemma 1 (Substitution lemma)**.** *If* $(\Omega : \eta, \psi)$ *is a lifted explanation graph, and* $X \in vars(\eta)$*, 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 $(\emptyset : \{\mathtt{false}\}, 0)$ 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 $(\emptyset : \{\mathtt{true}\}, \phi)$. This constitutes the informal proof of lemma 1.

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 10 (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* $Gr((\Omega : \eta, \psi))$*, is defined as follows*

$$
\begin{aligned}
Gr((\Omega : \eta, \psi)) \equiv{}& Gr(\Omega, \eta, \psi) \\
Gr(\Omega, \eta, (s,t)[\alpha_i : \psi_i]) \equiv{}& \begin{cases} (s,t)[\alpha_i : Gr(\Omega, \eta, \psi_i)], & \text{if } \eta \text{ is satisfiable and } t \notin \Omega \\ 0, & \text{otherwise} \end{cases} \\
Gr(\Omega, \eta, (s,t)[\alpha_i : \psi_i]) \equiv{}& \begin{cases} \bigvee_{c \in range(t,\eta)} (s,c)[\alpha_i : Gr(\Omega \setminus \{t\}, \eta[c/t], \psi_i[c/t])], & \text{if } \eta \text{ is satisfiable and } t \in \Omega \\ 0, & \text{otherwise} \end{cases} \\
Gr(\_,\_,0) \equiv{}& 0 \\
Gr(\_,\eta,1) \equiv{}& \begin{cases} 1, & \text{if } \eta \text{ is satisfiable} \\ 0, & \text{otherwise} \end{cases}
\end{aligned}
$$

In the above definition $\psi[c/t]$ represents the tree obtained by replacing every occurrence of t in the tree with c. The disjunct $(s,c)[\alpha_i : Gr(\Omega \setminus \{t\}, \eta[c/t], \psi_i[c/t])]$ 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. The key issue in defining these operations is to ensure the right order among the graph nodes (see criterion 3 of Def. 7). 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 $(\Omega : \eta, \psi) \wedge (\Omega' : \eta', \psi') \rightarrow (\emptyset : \{\texttt{false}\}, 0)$
- The $\vee$ operation simply returns the two inputs as outputs:

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

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

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

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

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 base cases are defined analogously).

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

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. $(s,t)<(s',t')$

$$(\Omega:\eta,\psi\oplus\psi')\rightarrow(\Omega:\eta,(s,t)[\alpha_i:\psi_i\oplus\psi'])$$

b. $(s',t')<(s,t)$

$$(\Omega:\eta,\psi\oplus\psi')\rightarrow(\Omega:\eta,(s',t')[\alpha_i':\psi\oplus\psi_i'])$$

c. $(s,t)=(s',t')$

$$(\Omega:\eta,\psi\oplus\psi')\rightarrow(\Omega:\eta,(s,t)[\alpha_i:\psi_i\oplus\psi_i'])$$

d. $(s,t)\not\sim(s',t')$

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

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

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

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

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 $l_1\leq l_2$ and we consider various cases of overlap as follows:

When $l_1 = l_2$ and $u_1 = u_2$

$$\begin{aligned}(\Omega : \eta, \psi \oplus \psi') \rightarrow (\Omega \cup \{t''\} : \eta \wedge l_1 - 1 < t'' \wedge t'' - 1 < u_1 \wedge t'' < t \wedge t'' < t',\\ &(s,t'')[\alpha_i :\\ &(\psi_i[t''/t] \oplus \psi_i'[t''/t'])\vee\\ &(\psi_i[t''/t] \oplus \psi')\vee\\ &(\psi_i'[t''/t'] \oplus \psi)])\end{aligned}$$

When $l_1 = l_2$ and $u_1 < u_2$ the result is

$$(\Omega : \eta \wedge t' - 1 < u_1, \psi \oplus \psi') \vee (\Omega : \eta \wedge u_1 < t', \psi \oplus \psi')$$

When $l_1 = l_2$ and $u_2 < u_1$ the result is

$$(\Omega : \eta \wedge t = t', \psi \oplus \psi') \vee (\Omega : \eta \wedge u_2 < t, \psi \oplus \psi')$$

When $l_1 < l_2$ and $u_1 = u_2$ the result is

$$(\Omega : \eta \wedge t = t', \psi \oplus \psi') \vee (\Omega : \eta \wedge t < l_2, \psi \oplus \psi')$$

When $l_1 < l_2$ and $u_1 < u_2$ the result is

$$(\Omega : \eta \wedge u_1 < t', \psi \oplus \psi') \vee (\Omega : \eta \wedge t < l_2 \wedge t' - 1 < u_1, \psi \oplus \psi') \vee (\Omega : \eta \wedge t = t', \psi \oplus \psi')$$

When $l_1 < l_2$ and $u_2 < u_1$ the result is

$$(\Omega : \eta \wedge u_2 < t, \psi \oplus \psi') \vee (\Omega : \eta \wedge t < l_2, \psi \oplus \psi') \vee (\Omega : \eta \wedge t = t', \psi \oplus \psi')$$

Lemma 2 (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 11 (Quantification)**.** *Operation $quantify((\Omega : \eta, \psi), X)$ changes a free variable $X \in vars(\eta)$ to a quantified variable. It is defined as*

$$quantify((\Omega : \eta, \psi), X) = (\Omega \cup \{X\} : \eta, \psi), \text{ if } X \in vars(\eta)$$

Lemma 3 (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 $\mathscr{P}$ 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 rangle $1..m$, 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.

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 12 (Frontier)**.** *Given a closed lifted explanation graph $(\Omega : \eta, \psi)$, the frontier of ψ w.r.t $X \in \Omega$ denoted $frontier_X(\psi)$ 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 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 13 (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 $\phi \in frontier_X(\psi)$ 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 $frontier_X(\psi)$ 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 the function $f(\sigma,\psi)$ where σ is a (possibly incomplete) mapping of variables in Ω to values in their domains.

Definition 14 (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 $range(t,\eta\sigma)=(l,u)$. Then*

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

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

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

In the above definition $\sigma[c/t]$ refers to a new partial mapping obtained by augmenting σ with the substitution $[c/t]$, $P(\widehat{\psi}_X)$ is the sum of the probabilities of all branches leading to a 1 leaf in $\widehat{\psi}_X$. The recurrences defining $f(\sigma,\psi)$, $g(\sigma,\psi)$ and $h(\sigma,\psi)$ are well-defined, and are computable.

Definition 15 (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 4 (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 (possibly implemented via tabling), a solution to the recurrence equations can be computed in polynomial time.

Theorem 5 (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, we use an instance of a *collective graphical model* (Sheldon and Dietterich 2011). In particular, 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. 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 a Markov model. The size of the lifted explanation graph, and the number of recurrences for this query is $O(k.t)$. When the recurrences are evaluated along three dimensions: time, total number of agents, and number of agents in state s, resulting in a time complexity of $O(n.k.t)$.

5 Related Work and Discussion

First-order graphical models (Poole 2003; Braz et al. 2005) 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* due to Poole (2003). 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* (Braz et al. 2005) and elimination by *counting formulas* (Milch et al. 2008).

Van den Broeck et al. (2011) presented a form of lifted inference that uses constrained CNF theories with positive and negative weight functions over predicates as input. 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.

Bellodi et al. (2014) present another approach to lifted inference for probabilistic logic programs. The idea is to convert a ProbLog program to parfactor representation and use a modified version of generalized counting first order variable elimination algorithm (Taghipour et al. 2013) 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. 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.

A Proofs

Lemma 2 (Correctness of "$\wedge$" and "$\vee$" operations)**.** *Let $(\Omega : \eta, \psi)$ and $(\Omega' : \eta', \psi')$ be two lifted explanation graphs with free variables $\{X_1, X_2 \dots, X_n\}$. 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 $\oplus \in \{\wedge, \vee\}$*

$$Gr(((\Omega : \eta, \psi) \oplus (\Omega' : \eta', \psi'))\sigma) = Gr((\Omega : \eta, \psi)\sigma) \oplus Gr((\Omega' : \eta', \psi')\sigma)$$

Proof. The proof is by structural induction.

Case 1: $Q(\Omega, \eta) \wedge Q(\Omega', \eta')$ is unsatisfiable.

Case 1.1: $\wedge$ operation

$Q(\Omega, \eta) \wedge Q(\Omega, \eta')$ is unsatisfiable, implies that for each $\sigma \in \Sigma$, either one or both of $Q(\Omega, \eta)\sigma$ and $Q(\Omega', \eta')\sigma$ are unsatisfiable. Either one or both of $\exists\Omega.\eta\sigma$ and $\exists\Omega'.\eta'\sigma$ are unsatisfiable. By definition 9, either one or both of the $(\Omega : \eta, \psi)\sigma$ and $(\Omega' : \eta', \psi')\sigma$ should be $(\emptyset : \{\texttt{false}\}, 0)$. By definition 10, we know that $Gr((\emptyset : \{\texttt{false}\}, 0)) = 0$. Therefore $Gr((\Omega : \eta, \psi)\sigma) \wedge Gr((\Omega' : \eta', \psi')\sigma) = 0$. The $\wedge$ operation in this case is defined as

$$(\Omega : \eta, \psi) \wedge (\Omega' : \eta', \psi') \rightarrow (\emptyset : \{\texttt{false}\}, 0)$$

Therefore, $Gr(((\Omega:\eta,\psi)\wedge(\Omega':\eta',\psi'))\sigma) = Gr((\Omega:\eta,\psi)\sigma)\wedge Gr((\Omega':\eta',\psi')\sigma)$.

Case 1.2: $\vee$ operation

Assume without loss of generality that $(\Omega':\eta',\psi')\sigma = (\emptyset:\{\texttt{false}\},0)$. Then by definition 10 $Gr((\Omega:\eta,\psi)\sigma)\vee Gr((\Omega':\eta',\psi')\sigma) = Gr((\Omega:\eta,\psi)\sigma)$. Since the definition of $\vee$ operation in this case is to backtrack and return both $(\Omega:\eta,\psi)$ and $(\Omega':\eta',\psi')$, under the substitution σ, $(\Omega':\eta',\psi')$ will be discarded. Therefore, $Gr(((\Omega:\eta,\psi)\vee(\Omega':\eta',\psi'))\sigma) = Gr((\Omega:\eta,\psi)\sigma)\vee Gr((\Omega':\eta',\psi')\sigma)$.

Case 2: $Q(\Omega,\eta)\wedge Q(\Omega',\eta')$ is satisfiable.

Case 2.1: $\wedge$ operation

The $\wedge$ operation is defined as $(\Omega:\eta,\psi)\wedge(\Omega':\eta',\psi')\rightarrow(\Omega\cup\Omega':\eta\wedge\eta',\psi\wedge\psi')$

Case 2.1.1: $(Q(\Omega,\eta)\wedge Q(\Omega',\eta'))\sigma$ is unsatisfiable

$(Q(\Omega,\eta)\wedge Q(\Omega',\eta'))\sigma$ is unsatisfiable, implies atleast one of $Q(\Omega,\eta)\sigma$ and $Q(\Omega',\eta)\sigma$ is unsatisfiable. Atleast one of $\exists\Omega.\eta\sigma$ and $\exists\Omega'.\eta'\sigma$ is unsatisfiable. By definition 9, atleast one of $(\Omega:\eta,\psi)\sigma$ and $(\Omega':\eta',\psi')\sigma$ is equal to $(\emptyset:\{\texttt{false}\},0)$. By definition 10, $Gr((\Omega:\eta,\psi)\sigma)\wedge Gr((\Omega':\eta',\psi')\sigma)=0$. Further, if one of $\exists\Omega.\eta\sigma$ and $\exists\Omega'.\eta'\sigma$ is unsatisfiable $\exists\Omega\cup\Omega'.\eta\wedge\eta'\sigma$ is also unsatisfiable. Therefore, by definition 9, $Gr((\Omega\cup\Omega':\eta\wedge\eta',\psi\wedge\psi')\sigma)=0$. Therefore $Gr(((\Omega:\eta,\psi)\wedge(\Omega':\eta',\psi'))\sigma) = Gr((\Omega:\eta,\psi)\sigma)\wedge Gr((\Omega':\eta',\psi')\sigma)$.

Case 2.1.1: $(Q(\Omega,\eta)\wedge Q(\Omega',\eta'))\sigma$ is satisfiable.

Case 2.1.1.1: $\psi=0$ (analogously $\psi'=0$).

By definition 10, $Gr((\Omega:\eta,\psi)\sigma)\wedge Gr((\Omega':\eta',\psi')\sigma)=0$. Based on the definition of $\wedge$ operation $(\Omega\cup\Omega':\eta\wedge\eta',\psi\wedge\psi')\sigma = (\Omega\cup\Omega':\eta\wedge\eta',0)\sigma$. By definition 10 $Gr((\Omega\cup\Omega':\eta\wedge\eta',0)\sigma)=0$. Therefore, $Gr(((\Omega:\eta,\psi)\wedge(\Omega':\eta',\psi'))\sigma) = Gr((\Omega:\eta,\psi)\sigma)\wedge Gr((\Omega':\eta',\psi')\sigma)$.

Case 2.1.1.2: $\psi=1$ (analogously $\psi'=1$).

By definition 10, $Gr((\Omega:\eta,\psi)\sigma)=1$. Therefore $Gr((\Omega:\eta,\psi)\sigma)\wedge Gr((\Omega':\eta',\psi')\sigma) = Gr((\Omega':\eta',\psi')\sigma)$. Based on the definition of $\wedge$ operation $(\Omega\cup\Omega':\eta\wedge\eta',\psi\wedge\psi') = (\Omega\cup\Omega':\eta\wedge\eta',\psi')$. By definition 9 $(\Omega\cup\Omega':\eta\wedge\eta',\psi')\sigma = (\Omega\cup\Omega':(\eta\wedge\eta')\sigma,\psi'\sigma)$ and $(\Omega':\eta',\psi')\sigma = (\Omega':\eta'\sigma,\psi'\sigma)$. We can claim that $Gr((\Omega':\eta'\sigma,\psi'\sigma)) = Gr((\Omega\cup\Omega':(\eta\wedge\eta')\sigma,\psi'\sigma))$ because, $range(t,\eta'\sigma) = range(t,(\eta\wedge\eta')\sigma)$ for any $t\in\Omega'$ Why? Because there exist no variables in common between $\eta\sigma$ and $\eta'\sigma$ and $\eta\sigma\wedge\eta'\sigma$ is satisfiable based on the assumptions.

Case 2.1.1.3 : Neither ψ nor ψ' is a leaf node and $(s,t)<(s',t')$ (analogously $(s',t')<(s,t)$).

Since $(Q(\Omega,\eta)\wedge Q(\Omega',\eta')\sigma)$ is satisfiable, we can conclude that $\eta\sigma$ and $\eta'\sigma$ are satisfiable. By definition 9, $Gr((\Omega:\eta,\psi)\sigma)\wedge Gr((\Omega':\eta',\psi')\sigma) = Gr(\Omega:\eta\sigma,\psi\sigma)\wedge Gr(\Omega':\eta'\sigma,\psi'\sigma)$. Let us consider the case where t is either a constant or a free variable. Since $(s,t)<(s',t')$ it implies that $(s,t\sigma)\prec(s',t'\sigma)$. By definition 10, $Gr(\Omega:\eta\sigma,\psi\sigma)\wedge Gr(\Omega':\eta'\sigma,\psi'\sigma) = Gr(\Omega,\eta\sigma,\psi\sigma)\wedge Gr(\Omega',\eta'\sigma,\psi'\sigma)$. Further $Gr(\Omega:\eta\sigma,\psi\sigma)\wedge Gr(\Omega':\eta'\sigma,\psi'\sigma) = (s,t\sigma)[\alpha_i: Gr(\Omega,\eta\sigma,\psi_i\sigma)\wedge Gr(\Omega',\eta'\sigma,\psi'\sigma)]$. Based on the definition of $\wedge$ operation

$(\Omega\cup\Omega' : \eta\wedge\eta', \psi\wedge\psi')\sigma = (\Omega\cup\Omega' : \eta\cup\eta', (s,t)[\alpha_i : \psi_i\wedge\psi'])\sigma$. Therefore, $Gr((\Omega\cup\Omega' : \eta\wedge\eta', \psi\wedge\psi')\sigma) = Gr(\Omega\cup\Omega', (\eta\wedge\eta')\sigma, (s,t)[\alpha_i : \psi_i\wedge\psi']\sigma) = (s,t\sigma)[\alpha_i : Gr(\Omega\cup\Omega', (\eta\wedge\eta')\sigma, (\psi_i\wedge\psi')\sigma)]$. Based on inductive hypothesis, $Gr(\Omega,\eta\sigma,\psi_i\sigma)\wedge Gr(\Omega',\eta'\sigma,\psi'\sigma) = Gr(\Omega\cup\Omega', (\eta\wedge\eta')\sigma, (\psi_i\wedge\psi')\sigma)$. Now consider the case when $t\in\Omega$. By definition 10 $Gr(\Omega : \eta,\psi)\sigma\wedge Gr(\Omega' : \eta',\psi')\sigma = Gr(\Omega,\eta\sigma,\psi\sigma)\wedge Gr(\Omega',\eta'\sigma,\psi'\sigma) = \bigvee_{c\in range(t,\eta\sigma)}(s,c)[\alpha_i : Gr(\Omega\setminus\{t\},\eta\sigma[c/t],\psi_i\sigma[c/t])\wedge Gr(\Omega,\eta'\sigma,\psi'\sigma)]$. Similarly, $Gr(\Omega\cup\Omega', (\eta\wedge\eta')\sigma, (s,t)[\alpha_i : \psi_i\wedge\psi']\sigma) = \bigvee_{c\in range(t,(\eta\wedge\eta')\sigma)}(s,c)[\alpha_i : Gr(\Omega\cup\Omega'\setminus\{t\}, (\eta\wedge\eta')\sigma[c/t], (\psi_i\wedge\psi')\sigma[c/t])]$. But $range(t,\eta\sigma) = range(t,(\eta\wedge\eta')\sigma)$ and based on inductive hypothesis, $Gr(\Omega\setminus\{t\},\eta\sigma[c/t],\psi_i\sigma[c/t])\wedge Gr(\Omega',\eta'\sigma,\psi'\sigma) = Gr(\Omega\cup\Omega'\setminus\{t\}, (\eta\wedge\eta')\sigma[c/t], (\psi_i\wedge\psi')\sigma[c/t])$

Case 2.1.1.4: Neither ψ nor ψ' is a leaf node and $(s,t) = (s',t')$.
Since the variables in the lifted explanation graphs are standardized apart, this implies that neither t nor t' is a bound variable. $Gr((\Omega : \eta,\psi)\sigma)\wedge Gr((\Omega' : \eta',\psi')\sigma) = (s,t\sigma)[\alpha_i : Gr(\Omega,\eta\sigma,\psi_i\sigma)\wedge Gr(\Omega',\eta'\sigma,\psi_i'\sigma)]$. Similarly, $Gr(\Omega\cup\Omega', (\eta\wedge\eta')\sigma, (s,t)[\alpha_i : \psi_i\wedge\psi_i']\sigma) = (s,t\sigma)[\alpha_i : Gr(\Omega\cup\Omega', (\eta\wedge\eta')\sigma, (\psi_i\wedge\psi_i')\sigma)]$. Based on inductive hypothesis, $Gr(\Omega,\eta\sigma,\psi_i\sigma)\wedge Gr(\Omega',\eta'\sigma,\psi_i'\sigma) = Gr(\Omega\cup\Omega', (\eta\wedge\eta')\sigma, (\psi_i\wedge\psi_i')\sigma)$.

Case 2.1.1.5: Neither ψ nor ψ' is a leaf node and $(s,t)\not\sim(s',t')$ and t is a free variable or a constant and t' is a free variable.
Consider the case where t and t' are free variables, or t is a constant and t' is a free variable. Based on σ exactly one of $(s,t\sigma) < (s,t'\sigma)$, $(s,t\sigma) = (s',t\sigma)$ and $(s,t'\sigma) < (s,t\sigma)$ will hold. According to the definition of $\wedge$ operation, three lifted explanation graphs are returned $(\Omega\cup\Omega' : \eta\wedge\eta'\wedge t<t', \psi\wedge\psi')$, $(\Omega\cup\Omega' : \eta\wedge\eta'\wedge t=t', \psi\wedge\psi')$, and $(\Omega\cup\Omega' : \eta\wedge\eta'\wedge t'<t, \psi\wedge\psi')$. Under the substitution σ only one will be retained. And the proof then proceeds as in case 2.1.1.3 or case 2.1.1.4.

Case 2.1.1.6: Neither ψ nor ψ' is a leaf node and $(s,t)\not\sim(s',t')$ and t is a free variable or a constant and t' is a bound variable. Consider $Gr((\Omega : \eta,\psi)\sigma)\wedge Gr((\Omega' : \eta',\psi')\sigma)$. This can be written as $(s,t\sigma)[\alpha_i : Gr(\Omega,\eta\sigma,\psi_i\sigma)]\wedge \bigvee_{c\in range(t',\eta'\sigma)}(s,c)[\alpha_i' : Gr(\Omega'\setminus\{t'\},\eta'\sigma[c/t'],\psi_i'\sigma[c/t'])]$. By using continuity of range we can rewrite it as

$$\begin{aligned}
&(s,t\sigma)[\alpha_i : Gr(\Omega,\eta\sigma,\psi_i\sigma)]\wedge \\
&\quad \bigvee_{c\in range(t',(\eta'\wedge t<t')\sigma)} (s,c)[\alpha_i' : Gr(\Omega'\setminus\{t'\},\eta'\sigma[c/t'],\psi_i'\sigma[c/t'])] \\
&\vee(s,t\sigma)[\alpha_i' : Gr(\Omega'\setminus\{t'\},\eta'\sigma[t\sigma/t'],\psi_i'\sigma[t\sigma/t'])] \\
&\quad \vee \bigvee_{c\in range(t',(\eta'\wedge t'<t)\sigma)} (s,c)[\alpha_i' : Gr(\Omega'\setminus\{t'\},\eta'\sigma[c/t'],\psi_i'\sigma[c/t'])]
\end{aligned}$$

By distributivity of $\wedge$ over $\vee$ for ground explanation graphs, we can rewrite it as

$$(s,t\sigma)[\alpha_i : Gr(\Omega, \eta\sigma, \psi_i\sigma)] \wedge \bigvee_{c \in range(t', (\eta' \wedge t < t')\sigma)} (s,c)[\alpha_i' : Gr(\Omega' \setminus \{t'\}, \eta'\sigma[c/t'], \psi_i'\sigma[c/t'$$

$$\vee (s,t\sigma)[\alpha_i : Gr(\Omega, \eta\sigma, \psi_i\sigma)] \wedge (s,t\sigma)[\alpha_i' : Gr(\Omega' \setminus \{t'\}, \eta'\sigma[t\sigma/t'], \psi_i'\sigma[t\sigma/t'])]$$

$$\vee (s,t\sigma)[\alpha_i : Gr(\Omega, \eta\sigma, \psi_i\sigma)] \bigvee_{c \in range(t', (\eta' \wedge t' < t)\sigma)} (s,c)[\alpha_i' : Gr(\Omega' \setminus \{t'\}, \eta'\sigma[c/t'], \psi_i'\sigma[c/t'$$

This can be re-written as

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

By inductive hypothesis this is equal to

$$\bigvee_{\varphi \in \{t < t', t = t', t' < t'\}} Gr((\Omega \cup \Omega' : \eta \wedge \eta' \wedge \varphi, \psi \wedge \psi')\sigma)$$

Case 2.1.1.7: Neither ψ nor ψ' is a leaf node and $(s,t) \not\sim (s',t')$ and t, t' are bound variables.

When $l_1 = l_2$ and $u_1 = u_2$, $Gr((\Omega : \eta, \psi)\sigma) \wedge Gr((\Omega' : \eta', \psi')\sigma)$ can be written as

$$\bigvee_{c \in [l_1, u_1]} (s,c)[\alpha_i : Gr((\Omega, \eta, \psi_i)\sigma[c/t])] \wedge \bigvee_{c' \in [l_2, u_2]} (s,c')[\alpha_i : Gr((\Omega', \eta', \psi_i')\sigma[c'/t'])]$$

Let the sequence of positive integers in the interval $[l_1, u_1]$ be $\langle l_1 = k_1, k_2, \ldots, k_n = u_1$. Then the above expression can be re-written as

$$\left((s,k_1)[\alpha_i : Gr((\Omega, \eta, \psi_i)\sigma[k_1/t])] \wedge \bigvee_{c' \in [k_1, k_n]} (s,c')[\alpha_i : Gr((\Omega', \eta', \psi_i')\sigma[c/t'])] \right) \vee$$

$$\left(((s,k_2)[\alpha_i : Gr((\Omega, \eta, \psi_i)\sigma[k_2/t])] \wedge \bigvee_{c' \in [k_1, k_n]} (s,c')[\alpha_i : Gr((\Omega', \eta', \psi_i')\sigma[c'/t'])] \right) \vee$$

$\ldots$

$$\left(((s,k_n)[\alpha_i : Gr((\Omega, \eta, \psi_i)\sigma[k_n/t])] \wedge \bigvee_{c' \in [k_1, k_n]} (s,c')[\alpha_i : Gr((\Omega', \eta', \psi_i')\sigma[c'/t'])] \right)$$

This can again be re-written as

$$\bigvee_{c\in[k_1,k_n]} \big((s,c)[\alpha_i : Gr((\Omega,\eta,\psi_i)\sigma[c/t])] \wedge (s,c)[\alpha_i : Gr((\Omega',\eta',\psi_i')\sigma[c/t'])]\big)$$

$$\bigvee_{d\in[k_1,k_{n-1}]} \left((s,d)[\alpha_i : Gr((\Omega,\eta,\psi_i)\sigma[d/t])] \wedge \bigvee_{e\in[d+1,k_n]} (s,e)[\alpha_i : Gr((\Omega',\eta',\psi_i')\sigma[e/t'])]\right)$$

$$\bigvee_{f\in[k_1,k_{n-1}]} \left((s,f)[\alpha_i : Gr((\Omega',\eta',\psi_i')\sigma[f/t])] \wedge \bigvee_{g\in[d+1,k_n]} (s,g)[\alpha_i : Gr((\Omega,\eta,\psi_i)\sigma[g/t'])]\right)$$

This is equivalent to

$$\begin{aligned}\bigvee_{c\in[k_1,k_{n-1}]} (s,c)[\alpha_i : (&Gr((\Omega,\eta,\psi_i)\sigma[c/t]) \wedge Gr((\Omega',\eta',\psi_i')\sigma[c/t']))\vee\\ &(Gr((\Omega,\eta,\psi_i)\sigma[c/t]) \wedge Gr((\Omega',\eta'\wedge t<t',(s,t')[\alpha_i : \psi_i'])\sigma[c/t]))\vee\\ &(Gr((\Omega',\eta',\psi_i')[t/t']\sigma[c/t]) \wedge Gr((\Omega,\eta\wedge t<t',(s,t')[\alpha_i : \psi_i])[t'/t]\sigma[c/t]))]\vee\\ &(s,k_n)[\alpha_i : (Gr((\Omega,\eta,\psi_i)\sigma[k_n/t]) \wedge Gr((\Omega',\eta',\psi_i')\sigma[k_n/t']))]\end{aligned}$$

By inductive hypothesis this is equivalent to

$$\begin{aligned}\bigvee_{c\in[k_1,k_n]} (s,c)[\alpha_i : &Gr((\Omega\cup\Omega',\eta\wedge\eta',\psi_i\wedge\psi_i')\sigma[c/t][c/t'])\vee\\ &Gr((\Omega\cup\Omega',\eta\wedge\eta'\wedge t<t',\psi_i\wedge(s,t')[\alpha_i : \psi_i'])\sigma[c/t])\vee\\ &Gr((\Omega\cup\Omega',\eta\wedge\eta'\wedge t<t',\psi_i'[t/t']\wedge(s,t')[\alpha_i : \psi_i[t'/t]])\sigma[c/t])\end{aligned}$$

But the definition of $\wedge$ operation is

$$\begin{aligned}(\Omega:\eta,\psi\wedge\psi')\wedge(\Omega':\eta',\psi') \to (\Omega\cup\Omega':\eta\wedge\eta'\wedge t<t',(s,t)[\alpha_i : &(\psi_i\wedge\psi_i'[t/t'])\vee\\ &(\psi_i\wedge\psi')\vee\\ &(\psi_i'[t/t']\wedge\psi[t'/t])])\end{aligned}$$

The grouding expansion given earlier can be rewritten as

$$\begin{aligned}Gr((\Omega\cup\Omega':\eta\wedge\eta'\wedge t<t',(s,t)[\alpha_i : &(\psi_i\wedge\psi'[t/t'])\vee\\ &(\psi_i\wedge\psi')\vee\\ &(\psi_i'[t/t']\wedge\psi[t'/t])]))\end{aligned}$$

Therefore the theorem is proved in this case.
The proof for remaining cases is analogous and straightforward, because based on the values of l_1, u_1, l_2 and u_2 we have disjuncts where the ranges of root variables are either identical or non-overlapping. Both of these cases have been proved already.

Case 2.2: Operation $\vee$

Case 2.2.1: When $Q(\Omega, \eta)$ is not identical to $Q(\Omega', \eta')$. The definition of $\vee$ operation returns several lifted explanation graphs. If $Q(\Omega, \eta)\sigma$ and $Q(\Omega', \eta')\sigma$ are both unsatisfiable then we can see that $Gr((\Omega : \eta, \psi)\sigma) \vee Gr((\Omega' : \eta', \psi')\sigma) = 0$. Further, all the lifted explanation graphs returned by the definition of $\vee$ have unsatisfiable constraints therefore, the theorem is proved in this case. When $Q(\Omega, \eta)\sigma$ is satisfiable but not $Q(\Omega', \eta')\sigma$ (or vice-versa), $Gr((\Omega : \eta, \psi)\sigma) \vee Gr((\Omega' : \eta', \psi')\sigma) = Gr((\Omega : \eta, \psi)\sigma)$. Based on the definition of $\vee$ operation, only the first lifted explanation graph has satisfiable constraint, so the theorem is proved. Same reasoning applies in the symmetric case.

Case 2.2.2: When $Q(\Omega, \eta)$ is identical to $Q(\Omega', \eta')$ and $Q(\Omega, \eta)\sigma$ is unsatisfiable, the proof of the theorem is trivial. So we consider the case where $Q(\Omega, \eta)\sigma$ and $Q(\Omega', \eta')\sigma$ are both satisfiable and $Q(\Omega, \eta)$ may or may not be identical to $Q(\Omega', \eta')$.

Case 2.2.2.1: When $\psi = 0$ (analogously $\psi' = 0$). Here $Gr((\Omega : \eta, \psi)\sigma) \vee Gr((\Omega' : \eta', \psi')\sigma) = Gr((\Omega' : \eta', \psi')\sigma)$. Similarly $Gr((\Omega \cup \Omega' : \eta \wedge \eta', \psi')\sigma) = Gr((\Omega' : \eta', \psi')\sigma)$. Therefore the theorem is proved.

Case 2.2.2.2: When $\psi = 1$ (analogously $\psi' = 1$). Here $Gr((\Omega : \eta, \psi)\sigma) \vee Gr((\Omega' : \eta', \psi')\sigma) = 1$. Similarly $Gr((\Omega \cup \Omega' : \eta \wedge \eta', 1)\sigma) = 1$. Therefore, the theorem is proved.

Case 2.2.2.3: Neither ψ nor ψ' is a leaf node and $(s,t) < (s,t')$ (analogously $(s',t') < (s,t)$). Proof is analogous to Case 2.1.1.3.

Case 2.2.2.4: Neither ψ nor ψ' is a leaf node and $(s,t) = (s',t')$. Proof is analogous to Case 2.1.1.4.

Case 2.2.2.5: Neither ψ nor ψ' is a leaf node and $(s,t) \not\sim (s',t')$ and t is a free variable or a constant and t' is a free variable. Proof is analogous to Case 2.1.1.5.

Case 2.2.2.6: Neither ψ nor ψ' is a leaf node and $(s,t) \not\sim (s',t')$ and t is a free variable or a constant and t' is a bound variable. Proof is analogous to Case 2.1.1.6.

Case 2.2.2.7: Neither ψ nor ψ' is a leaf node and $(s,t) \not\sim (s',t')$ and t, t' are bound variables. Proof is analogous to Case 2.1.1.7.

□

Lemma 3 (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*

$$Gr(quantify((\Omega : \eta, \psi), X)\sigma_{-X}) = \bigvee_{\sigma \in \Sigma} Gr((\Omega : \eta, \psi)\sigma)$$

Proof. Let us first consider the case when the root is (s, X) for some switch s in ψ. $quantify((\Omega : \eta, \psi), X) = (\Omega \cup \{X\} : \eta, \psi)$. If $\eta\sigma_{-X}$ is unsatisfiable, then $Gr(quantify((\Omega : \eta, \psi), X)\sigma_{-X}) = 0$.

Next, $\bigvee_{\sigma\in\Sigma} Gr((\Omega:\eta,\psi)\sigma[c/X]) = 0$ since $\eta\sigma$ is also unsatisfiable for any σ. On the other hand if $\eta\sigma_{-X}$ is satisfiable,

$$\begin{aligned} Gr(quantify((\Omega:\eta,\psi),X)\sigma_X) &= Gr((\Omega\cup\{X\}:\eta,\psi)\sigma_{-X}) \\ &= Gr((\Omega\cup\{X\}:\eta\sigma_{-X},\psi\sigma_{-X})) \\ &= \bigvee_{c\in range(X,\eta\sigma_{-X})} (s,c)[\alpha_i : Gr(\Omega\setminus\{X\},\eta\sigma_{-X}[c/X],\psi_i\sigma_{-X}[c/X])] \end{aligned}$$

Next,

$$\bigvee_{\sigma\in\Sigma} Gr((\Omega:\eta,\psi)\sigma) = \bigvee_{c\in\sigma_X(\eta\sigma_{-X})} (s,c)[\alpha_i : Gr(\Omega\setminus\{X\},\eta\sigma_{-X}[c/X],\psi_i\sigma_{-X}[c/X])]$$

By using continuity of range, $range(X,\eta\sigma_{-X}) = \sigma_X(\eta\sigma_{-X})$. Therefore the theorem is proved in this case. Now consider the case where X doesn't occur in the root of the lifted explanation graph. Since, the lifted explanation graph is well-structured, there is subtree ψ' in ψ such that the root of ψ' contains X and all occurrences of X are within ψ'. If we remove the subtree ψ' from ψ, then $Gr(quantify((\Omega:\eta,\psi),X)\sigma_{-X}) = \bigvee_{\sigma\in\Sigma} Gr((\Omega:\eta,\psi)\sigma)$ since all the disjuncts on the right hand side will be identical to each other and to the ground explanation tree on the left hand side. Therefore, we need only show that the grounding of the subtree ψ' when X is a quantified variable is same as $\bigvee_{\sigma\in\Sigma} Gr((\Omega,\eta\sigma,\psi'\sigma)$ which we already showed. ☐

Theorem 4 (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 $prob((\Omega:\eta,\psi)) = prob(\phi)$.*

Proof. Consider the following modification of the grounding algorithm for lifted explanation graphs. An extra argument is added to $Gr(\Omega,\eta,\psi)$ to make it $Gr(\Omega,\eta,\psi,\sigma)$. Whenever a variable is substituted by a value from its domain, σ is augmented to record the substitution. Further the set Ω and the constraint formula η are not altered when recursively grounding subtrees. Rather, $\eta\sigma$ is tested for satisfiability and $t\sigma$ is tested for membership in Ω to determine if a node contains bound variable. The grounding of a lifted explanation graph $(\Omega:\eta,\psi)$ is given by $Gr(\Omega,\eta,\psi,\{\})$. It is easy to see that the ground explanation tree produced by this modified procedure is same as that produced by the procedure given in definition 10.

We will prove that if $Gr(\Omega,\eta,\psi,\sigma) = \phi$, then $prob(\phi) = f(\sigma,\psi)$. We prove this using structural induction based on the structure of ψ.

Case 1: If ψ is a 0 leaf node, then $Gr(\Omega,\eta,0,\sigma) = 0$. Therefore $prob(\phi) = f(\sigma,\psi) = 0$.

Case 2: If ψ is a 1 leaf node, and $\eta\sigma$ is satisfiable, then $Gr(\Omega,\eta,\psi,\sigma) = 1$ and $prob(\phi) = f(\sigma,\psi)$. On the other hand if $\eta\sigma$ is not satisfiable, then $Gr(\Omega,\eta,\psi,\sigma) = 0$ and $prob(\phi) = f(\sigma,\psi)$.

Case 3: If $\psi = (s,t)[\alpha_i : \psi_i]$ and $t\sigma \notin \Omega$, and $\eta\sigma$ is satisfiable, then $\phi = (s,t\sigma)[\alpha_i : Gr(\Omega,\eta,\psi_i,\sigma)]$. Therefore, $prob(\phi) = \sum_{\alpha_i\in D_s} \pi_s(\alpha_i)\cdot prob(Gr(\Omega,\eta,\psi_i,\sigma))$. But $f(\sigma,\psi) = \sum_{\alpha_i\in D_s} \pi_s(\alpha_i)\cdot f(\sigma,\psi_i)$. Therefore, by inductive hypothesis the theorem is proved in this case. If $\eta\sigma$ is unsatisfiable, then $\phi = 0$, therefore $prob(\phi) = f(\sigma,\psi)$.

Case 4: If $\psi = (s,t)[\alpha_i : \psi_i]$ and $t\sigma \in \Omega$ and $\eta\sigma$ is satisfiable. In this case ϕ is defined as the disjunction of the ground trees $(s,c)[\alpha_i : Gr(\Omega,\eta,\psi_i,\sigma[c/t])]$ where $c \in range(t,\eta\sigma)$. Let us order the grounding trees in the increasing order of the value c. Given two trees $\phi_{(s,c)}$ and $\phi_{(s,c+1)}$ corresponding to values $c, c+1 \in range(t,\eta\sigma)$, the $\vee$ operation on ground trees, would recursively perform disjunction of the $\phi_{(s,c+1)}$ with the subtrees in the following set

$$Fr = \{\phi' \mid \phi' \text{ is a maximal subtree of } \phi_{(s,c)} \text{ without } c \text{ as instance argument of any node}\}$$

The set Fr contains ground trees corresponding to the trees in $frontier_t(\psi)$ and possibly 0 leaves. Since we assumed that frontier subsumption property is satisfied, for every $\phi' \in Fr$ that is not a 0 leaf, it holds that every explanation in $\phi_{(s,c+1)}$ contains a subexplanation in ϕ'. Therefore, $\phi_{(s,c)} \vee \phi_{(s,c+1)}$, can be computed equivalently as $\phi_{(s,c)} \vee (\neg\widehat{\psi}_t[c/t] \wedge \phi_{(s,c+1)})$. Since $\neg\widehat{\psi}_t[c/t]$ contains only internal nodes with instance argument c, the explanations of $\neg\widehat{\psi}_t[c/t]$ are independent of explanations in $\phi_{(s,c+1)}$. Further, the explanations of $\phi_{(s,c)}$ are mutually exclusive with explanations in $\neg\widehat{\psi}_t[c/t]$. Therefore the probability $prob(\phi_{(s,c)} \vee \phi_{(s,c+1)})$ can be computed as $prob(\phi_{(s,c)}) + (1 - prob(\widehat{\psi}_t[c/t])) \cdot prob(\psi_{(s,c+1)})$. The probability of the complete disjunction $\bigvee_{c \in range(t,\eta\sigma)}(s,c)[\alpha_i : Gr(\Omega,\eta,\psi_i,\sigma[c/t])]$ is obtained by the expression

$$\begin{aligned}
&prob(\phi_{(s,l)}) + (1 - prob(\widehat{\psi}_t[l/t])) \times (\\
&\qquad prob(\phi_{(s,l+1)}) + (1 - prob(\widehat{\psi}_t[l+1/t])) \times (\\
&\qquad\qquad \cdots \\
&\qquad\qquad (1 - prob(\widehat{\psi}_t[u-1/t])) \times prob(\phi_{(s,u)})))
\end{aligned}$$

Now consider $f(\sigma,\psi)$ for the same ψ, $f(\sigma,\psi) = h(\sigma[l/t],\psi)$. The expansion of $h(\sigma[l/t],\psi)$ is as follows

$$\begin{aligned}
&g(\sigma[l/t],\psi) + (1 - prob(\widehat{\psi}_t[l/t])) \times (\\
&\qquad g(\sigma[l+1/t],\psi) + (1 - prob(\widehat{\psi}_t[l+1/t])) \times (\\
&\qquad\qquad \cdots \\
&\qquad\qquad (1 - prob(\widehat{\psi}_t[u-1/t]) \times g(\sigma[u/t],\psi)
\end{aligned}$$

For a given ground tree $\phi_{(s,c)}$, $prob(\phi_{(s,c)}) = \sum_{\alpha_i \in D_s} \pi_s(\alpha_i) \cdot prob(Gr(\Omega,\eta,\psi_i,\sigma[c/t]))$. Similarly $g(\sigma[c/t],\psi) = \sum_{\alpha_i \in D_s} \pi_s(\alpha_i) \cdot f(\sigma[c/t],\psi_i)$. But by inductive hypothesis, $Prob(Gr(\Omega,\eta,\psi_i,\sigma[c/t])) = f(\sigma[c/t],\psi_i)$. Therefore, $prob(\phi_{(s,c)}) = g(\sigma[c/t],\psi)$. Therefore $prob(\phi) = f(\sigma,\psi)$. When, $\eta\sigma$ is not satisfiable, $prob(\phi) = f(\sigma,\psi) = 0$. Therefore, the theorem is proved.

□

References

[LLP 1997] 1997. Efficient verification of real-time systems: Compact data structure and state-space reduction. In *In IEEE RTSS97*. 14–24.

[Bellodi et al. 2014] BELLODI, E., LAMMA, E., RIGUZZI, F., COSTA, V. S., AND ZESE, R. 2014. Lifted variable elimination for probabilistic logic programming. *TPLP 14,* 4-5, 681–695.

[Braz et al. 2005] BRAZ, R. D. S., AMIR, E., AND ROTH, D. 2005. Lifted first-order probabilistic inference. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*. 1319–1325.

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

[Darwiche 2001] DARWICHE, A. 2001. On the tractable counting of theory models and its application to truth maintenance and belief revision. *Journal of Applied Non-Classical Logics 11,* 1-2, 11–34.

[Darwiche 2011] DARWICHE, A. 2011. SDD: A new canonical representation of propositional knowledge bases. In *International Joint Conference on Artificial Intelligence*. 819–826.

[De Raedt et al. 2007] DE RAEDT, L., KIMMIG, A., AND TOIVONEN, H. 2007. ProbLog: A probabilistic prolog and its application in link discovery. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*. Vol. 7. 2462–2467.

[Dries et al. 2015] DRIES, A., KIMMIG, A., MEERT, W., RENKENS, J., DEN BROECK, G. V., VLASSELAER, J., AND RAEDT, L. D. 2015. Problog2: Probabilistic logic programming. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD*. 312–315.

[Kisynski 2010] KISYNSKI, J. J. 2010. Aggregation and constraint processing in lifted probabilistic inference. Ph.D. thesis, The University of British Columbia (Vancouver.

[Milch et al. 2008] MILCH, B., ZETTLEMOYER, L. S., KERSTING, K., HAIMES, M., AND KAELBLING, L. P. 2008. Lifted probabilistic inference with counting formulas. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*. 1062–1068.

[Miné 2006] MINÉ, A. 2006. The octagon abstract domain. *Higher-Order and Symbolic Computation 19,* 1, 31–100.

[Poole 2003] POOLE, D. 2003. First-order probabilistic inference. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*. Vol. 3. 985–991.

[Riguzzi and Swift 2011] RIGUZZI, F. AND SWIFT, T. 2011. The PITA system: Tabling and answer subsumption for reasoning under uncertainty. *Theory and Practice of Logic Programming 11,* 4-5, 433–449.

[Sato and Kameya 1997] SATO, T. AND KAMEYA, Y. 1997. PRISM: a language for symbolic-statistical modeling. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*. Vol. 97. 1330–1339.

[Sato and Kameya 2001] SATO, T. AND KAMEYA, Y. 2001. Parameter learning of logic programs for symbolic-statistical modeling. *Journal of Artificial Intelligence Research*, 391–454.

[Sheldon and Dietterich 2011] SHELDON, D. R. AND DIETTERICH, T. G. 2011. Collective graphical models. In *Advances in Neural Information Processing Systems 24*. 1161–1169.

[Srinivasan et al. 1990] SRINIVASAN, A., KAM, T., MALIK, S., AND BRAYTON, R. K. 1990. Algorithms for discrete function manipulation. In *International Conference on Computer-Aided Design, ICCAD*. 92–95.

[Taghipour et al. 2013] TAGHIPOUR, N., FIERENS, D., DAVIS, J., AND BLOCKEEL, H. 2013. Lifted variable elimination: Decoupling the operators from the constraint language. *J. Artif. Intell. Res. (JAIR) 47*, 393–439.

[Van den Broeck et al. 2014] VAN DEN BROECK, G., MEERT, W., AND DARWICHE, A. 2014. Skolemization for weighted first-order model counting. In *Principles of Knowledge Representation and Reasoning*.

[Van den Broeck et al. 2011] VAN DEN BROECK, G., TAGHIPOUR, N., MEERT, W., DAVIS, J., AND DE RAEDT, L. 2011. Lifted probabilistic inference by first-order knowledge compilation. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence*. 2178–2185.

[Yovine 1998] YOVINE, S. 1998. Model-checking timed automata. In *Embedded Systems*. Number 1494 in LNCS. 114–152.