# A Survey of Lifted Inference Approaches for Probabilistic Logic Programming under the Distribution Semantics

Fabrizio Riguzzi[a,*], Elena Bellodi[b], Riccardo Zese[b], Giuseppe Cota[b], Evelina Lamma[b]

[a] *Dipartimento di Matematica e Informatica – Università di Ferrara*
*Via Saragat 1, 44122, Ferrara, Italy*
[b] *Dipartimento di Ingegneria – Università di Ferrara*
*Via Saragat 1, 44122, Ferrara, Italy*

**Abstract**

Lifted inference aims at answering queries from statistical relational models by reasoning on populations of individuals as a whole instead of considering each individual singularly. Since the initial proposal by David Poole in 2003, many lifted inference techniques have appeared, by lifting different algorithms or using approximation involving different kinds of models, including parfactor graphs and Markov Logic Networks. Very recently lifted inference was applied to Probabilistic Logic Programming (PLP) under the distribution semantics, with proposals such as $LP^2$ and Weighted First-Order Model Counting (WFOMC). Moreover, techniques for dealing with aggregation parfactors can be directly applied to PLP. In this paper we survey these approaches and present an experimental comparison on five models. The results show that WFOMC outperforms the other approaches, being able to exploit more symmetries.

*Keywords:* Probabilistic Logic Programming, Lifted Inference, Variable Elimination, Distribution Semantics, ProbLog, Statistical Relational Artificial Intelligence

## 1. Introduction

Statistical relational models [1, 2] describe domains with many individual entities connected by uncertain relations. Reasoning with models of the real world is often very costly due to the complexity of the models. However, sometimes the cost of reasoning can be reduced by exploiting symmetries in the model. This is the task of "lifted" inference, that answers queries by reasoning on populations of individuals as a whole instead of considering each individual

*Corresponding author
*Email addresses:* fabrizio.riguzzi@unife.it (Fabrizio Riguzzi), elena.bellodi@unife.it (Elena Bellodi), riccardo.zese@unife.it (Riccardo Zese), giuseppe.cota@unife.it (Giuseppe Cota), evelina.lamma@unife.it (Evelina Lamma)

singularly. The exploitation of the symmetries in the model can significantly speed up inference.

Lifted inference was initially proposed by David Poole in 2003 [3]. Since then, many techniques have appeared, lifting algorithms such as variable elimination and belief propagation, using approximation and dealing with models such as parfactor graphs and Markov Logic networks [4, 5, 6].

Lifted inference was applied to Probabilistic Logic Programming (PLP) only very recently. The first work is [7], where the authors describe the Prolog Factor Language (PFL), a representation in Prolog of first-order probabilistic factor models. The authors also present an implementation of lifted variable elimination and lifted belief propagation for PFL.

In PLP, most languages are based on the distribution semantics [8], such as Probabilistic Horn Abduction [9], PRISM [10], Independent Choice Logic [11], Logic Programs with Annotated Disjunctions [12], and ProbLog [13, 14]. Applying lifted inference to PLP languages under the distribution semantics (PLPDS) is problematic because the conclusions of different rules are combined with noisy-OR that requires aggregations at the lifted level when existential variables are present. For example, consider the following ProbLog program from [15]:

```
p :: famous(Y).
popular(X) :- friends(X, Y), famous(Y).
```

where $p$ is a real value corresponding with the probability of the probabilistic fact. In this case $P(\texttt{popular(john)}) = 1 - (1-p)^m$ where $m$ is the number of friends of `john`. This is because the body contains a variable not appearing in the head, that is thus existentially quantified. A grounding of the atom in the head of this clause represents the noisy-OR (without *leak probability*) of a number of ground bodies. In this case we do not need to know the identities of these friends, we just need to know how many there are. Hence, we need not to ground the clauses.

An exhaustive survey about lifted inference was proposed in [16]. However its focus is on Statistical Relational Learning and probabilistic graphical models techniques in general and does not handle specifically existential variables and aggregation.

The first works applying lifted inference directly to PLPDS appeared in 2014. In [17] the authors proposed $\text{LP}^2$ (for Lifted Probabilistic Logic Programming) that answers queries to ProbLog by translating the program into PFL and using an extended GC-FOVE lifted variable elimination algorithm.

Weighted First Order Model Counting (WFOMC) [18] instead uses a Skolemization algorithm for model counting problems that eliminates existential quantifiers from a first-order logic theory without changing its weighted model count. As such, it can be applied to PLPDS.

Aggregation is also treated in [19] where the authors proposed an aggregation operator for first directed first-order models that is independent of the sizes of the populations, in order to handle contexts in which a parent random variable is parameterized by logical variables that are not present in a child random variable.

In this paper, we survey these three proposals and experimentally evaluate them. The results show that inference time linearly increases with the number of individuals of the domain for approaches exploiting lifted variable elimination, while it is constant in case of WFOMC, thus indicating that the latter is able to lift a larger portion of the model.

The paper is organized as follows. Section 2 introduces preliminaries regarding ProbLog, PFL, Causal Independence Variable Elimination, and GC-FOVE. Section 3 presents LP$^2$ and shows the translation of ProbLog into PFL. Section 4 illustrates the use of aggregation parfactors for ProbLog. Section 5 describes WFOMC. Section 6 discusses how to apply these algorithms to non-tight logic programs. Section 7 reports the experiments performed and Section 8 concludes the paper.

## 2. Preliminaries

### *2.1. Notation*

Lifted inference techniques exploit concepts from relational logic and probabilistic theory. Unfortunately these two branches of mathematics sometimes use the same term to indicate different concepts. For example the word "variable" means logical variable in the context of relational logic, whereas it means random variable in the field of probabilistic theory. In order to avoid confusion, we use different fonts to represent different meanings. Table 1 shows the notation used throughout the paper.

| Concept | Notation |
|---|---|
| Logical variable | Typewriter upper case letters $\mathtt{X}, \mathtt{Y}, \ldots$ |
| Vector of logical variables | Typewriter bold case letters $\mathbf{X}$, $\mathbf{Y}$, ... |
| Constant | Typewriter lower case letters $\mathtt{x}$, $\mathtt{y}$, ... |
| Factor | Italic upper case letters or (if the context is clear) Greek letters $X$, $\phi$,... |
| Logical atom/predicate symbol, random variable (RV) | Italic upper case letters $X$, $Y$, ... |
| Value assigned to RV | Italic lower case letters $x$, $y$, ... |
| Vector of RVs | Bold italic upper case letters $\boldsymbol{X}$, $\boldsymbol{Y}$, ... |
| Value assigned to vector of (parameterized) RVs | Bold italic lower case letters $\boldsymbol{x}$, $\boldsymbol{y}$, ... |
| Parameterized random variable (PRV) or parfactor | Italic sans serif upper case letters $\mathsf{X}$, $\mathsf{Y}$, ... |
| Vector of PRVs | Bold italic sans serif upper case letters $\boldsymbol{\mathsf{X}}$, $\boldsymbol{\mathsf{Y}}$, ... |
| Set of constraints | Calligraphic $\mathcal{C}$ |
| Code | Typewriter |

Table 1: Notation used in this paper.

### *2.2. ProbLog*

ProbLog [13, 14] is a PLP language with a simple syntax and can be considered as the prototype of PLPDS. A ProbLog program consists of a set of

rules (a normal logic program) plus a set of *ground probabilistic facts*. Ground probabilistic facts are facts annotated with a real value $p$ in the interval $[0, 1]$ and are written as $p :: F$. A *probabilistic atom* is an atom that depends on probabilistic facts.

ProbLog provides syntactic sugar for the compact definition of a set of probabilistic facts with a single clause. For instance, if a set of ground probabilistic facts has the same probability $p$, it can be defined intensionally through the syntax $p :: F(\mathtt{X}_1, \mathtt{X}_2 \ldots, \mathtt{X}_n) :\!\!-\ B$, where $F(\mathtt{X}_1, \mathtt{X}_2, \ldots, \mathtt{X}_n)$ is the non-ground atom that identifies the set of probabilistic ground facts, and $B$ is a conjunction of non-probabilistic atoms, as shown in Example 1. Such rules must be range-restricted: all variables in the head of a rule should also appear in a positive literal in the body.

**Example 1 (Running example).** *The following program is inspired by the workshop attributes problem of [5]. It models the organization of a workshop where a number of people have been invited.* `series` *indicates whether the workshop is successful enough to start a series of related meetings while* `attends(P)` *indicates whether person* `P` *will attend the workshop. Note that all rules are range-restricted, i.e., all variables in the head appear also in a positive literal in the body.*

```
series :- self.
series :- attends(P).
attends(P) :- at(P,A).
0.1::self.
0.3::at(P,A) :- person(P), attribute(A).
```

*A workshop becomes a series either because of its own merits with a 10% probability (represented by the probabilistic fact* `self`*) or because people attend. People attend the workshop depending on the workshop's attributes such as location, date, fame of the organizers, etc (modeled by the probabilistic fact* `at(P,A)`*). The probabilistic fact* `at(P,A)` *represents whether person* `P` *attends because of attribute* `A`*. Note that the last statement corresponds to a set of ground probabilistic facts, one for each person* `P` *and attribute* `A`*. For the sake of brevity we omit the (non-probabilistic) facts describing the* `person/1` *and* `attribute/1` *predicates.*

A ProbLog program specifies a probability distribution over normal logic programs called *worlds* as defined by the distribution semantics. In this work, we consider the semantics in the case of no function symbols, and assume all worlds have a *two-valued* well-founded model. For the semantics with function symbols see [20].

An *atomic choice* specifies whether a ground probabilistic fact $p_i :: F_i$ is included in a world (with probability $p_i$) or not (with probability $1 - p_i$). A *total choice* $C$ contains an atomic choice for each ground probabilistic fact and identifies a *world*, i.e. a normal logic program $W = F \cup R$, where $F$ is the set of facts to be included according to $C$ and $R$ denotes the rules in the ProbLog

program. Let $\mathcal{W}$ be the set of all possible worlds. The probability of a world is the product of the probabilities of the individual atomic choices contained in the corresponding total choice $C$, $P(C) = \Pi_i q_i$ where $q_i = p_i$ if the fact is chosen and $q_i = 1 - p_i$ if the fact is not chosen. This is correct under the assumption that atomic choices are pairwise independent. Given a query (a ground atom) $Q$, the conditional probability of $Q$ given a world $W$ $P(Q|W)$ is 1 if $Q$ is true in the well-founded model of $W$ and 0 otherwise. Hence, the probability of a query is $P(Q) = \sum_{W \in \mathcal{W}} P(Q, W) = \sum_{W \in \mathcal{W}} P(Q|W)P(W) = \sum_{W \in \mathcal{W}: W \models Q} P(W)$.

### *2.3. The Prolog Factor Language*

The Prolog Factor Language (PFL) [7] is an extension of Prolog for representing first-order probabilistic models. It already includes a tool for performing lifted inference such as the GC-FOVE algorithm [21], that is used in the experimental evaluation in Section 7.

Most graphical models such as Bayesian and Markov Networks can concisely represent a joint distribution by encoding it as a set of factors. The probability of a set of variables $\boldsymbol{X}$ taking value $\boldsymbol{x}$ can be expressed as the product of $n$ factors as:

$$P(\boldsymbol{X} = \boldsymbol{x}) = \frac{\prod_{i=1,\ldots,n} \phi_i(\boldsymbol{x}_i)}{Z}$$

where $\boldsymbol{x}_i$ is a sub-vector of $\boldsymbol{x}$ on which the $i$-th factor depends and $Z$ is a normalization constant (i.e. $Z = \sum_{\boldsymbol{x}} \prod_{i=1,\ldots,n} \phi_i(\boldsymbol{x}_i)$). For example, in Bayesian networks there is a factor for each variable $X_i$ that is a function of the variable and its parents $X_j \ldots X_k$, such that $\phi_i(X_i, X_j, \ldots, X_k) = P(X_i|X_j \ldots X_k)$ and $Z = 1$. Often, in a graphical model, the same factors appear repeatedly in the network, thus we can parameterize these factors in order to simplify the representation.

A *parameterized random variable* (PRV) represents a set of random variables, one for each possible ground substitution to all of its parameters. A *parametric factor* or *parfactor* [22] is a triple $\langle \mathcal{C}, \mathcal{V}, F \rangle$ where $\mathcal{C}$ is a set of inequality constraints on parameters (logical variables), $\mathcal{V}$ is a set of parameterized random variables and $F$ is a factor that is a function from the Cartesian product of ranges of parameterized random variables in $\mathcal{V}$ to real values. A parfactor is also represented as $F(\mathcal{V})|\mathcal{C}$ or $F(\mathcal{V})$ if there are no constraints. A PRV $\mathsf{V}$ is of the form $\mathsf{V}|\mathcal{C}$, where $\mathsf{V} = P(\mathtt{X_1}, \ldots, \mathtt{X_n})$ is a non-ground atom and $\mathcal{C}$ is a set of constraints on logical variables $\mathbf{X} = \{\mathtt{X_1}, \ldots, \mathtt{X_n}\}$. Each PRV represents the set of random variables $\{P(\mathbf{x})|\mathbf{x} \in \mathcal{C}\}$, where $\mathbf{x}$ is the tuple of constants $(\mathtt{x_1}, \ldots, \mathtt{x_n})$. Given a PRV $\mathsf{V}$, we use $RV(\mathsf{V})$ to denote the set of random variables it represents. Each ground atom is associated with one random variable, which can take any value in $range(\mathsf{V})$.

The Prolog Factor Language [7] extends Prolog to support probabilistic reasoning with parametric factors. A PFL factor is a parfactor[1] of the form

[1] The terminology is a bit unfortunate but we preferred to stick with the one most widely used.

*Type* $\boldsymbol{F}$ ; $\phi$ ; $\mathcal{C}$, where *Type* refers to the type of the network over which the parfactor is defined (*bayes* for directed networks or *markov* for undirected ones); $\boldsymbol{F}$ is a sequence of Prolog goals each defining a set of random variables under the constraints in $\mathcal{C}$ (the arguments of the factor). Let us call $\mathtt{L}$ the set of all logical variables in $\boldsymbol{F}$, then $\mathcal{C}$ is a list of Prolog goals that impose bindings on $\mathtt{L}$ (the successful substitutions for the goals in $\mathcal{C}$ are the valid values for the variables in $\mathtt{L}$). $\phi$ is the table defining the factor in the form of a list of real values. By default, all random variables are Boolean but a different domain may be defined. Each parfactor represents the set of its groundings. To ground a parfactor, all variables of $\mathtt{L}$ are replaced with the values permitted by constraints in $\mathcal{C}$. The set of ground factors defines a factorization of the joint probability distribution over all random variables.

**Example 2 (PFL Program).** *A version of the workshop attributes problem presented in Example 1 can be modeled by a PFL program such as*

```
bayes series, attends(P) ; [0.51, 0.49, 0.49, 0.51] ;
  [person(P)].
bayes attends(P), at(P,A) ; [0.7, 0.3, 0.3, 0.7] ;
  [person(P),attribute(A)].
```

*The first PFL factor has the Boolean random variables* `series` *and* `attends(P)` *as arguments,* `[0.51,0.49,0.49,0.51]` *as table and* `[person(P)]` *as constraint.*

*This model is not equivalent to the one of Example 1, but it corresponds to a ProbLog program that has only the second and the third clause of Example 1. Equivalent models will be given in Examples 4 and 7.*

### *2.4. Variable Elimination*

Variable Elimination (VE) [23, 24] is an algorithm for probabilistic inference from graphical models. VE takes as input a set of factors $\mathcal{F}$, an elimination order $\rho$, a query variable $X$ and a list $\boldsymbol{y}$ of observed values. After setting the observed variables in all factors to their corresponding observed values, VE eliminates the random variables from the factors one by one until only the query variable $X$ remains. This is done by selecting the first variable $Z$ from the elimination order $\rho$ and then calling SUM-OUT that eliminates $Z$ by first multiplying all the factors that include $Z$ into a single factor and summing out $Z$ from the newly constructed factor. This procedure is repeated until $\rho$ becomes empty. In the final step, VE multiplies together the factors of $\mathcal{F}$ obtaining a new factor $\gamma$ that is normalized as $\gamma(x)/\sum_{x'}\gamma(x')$ to give the posterior probability.

In many cases, we need to represent factors where a Boolean variable $X$ with parents $\boldsymbol{Y}$ is true if any of the $Y_i$ is true. In practice, each parent $Y_i$ has a noisy inhibitor that independently blocks or activates $Y_i$, so $X$ is true if either **any** of the causes $Y_i$ holds true *and* is not inhibited. This is called *noisy-OR gate*. Handling this kind of factor is a non trivial problem.

A noisy-OR factor can be expressed as a combination of factors by introducing intermediate variables that represent the effect of each cause *given the*

*inhibitor*. For example, if $X$ has two causes $Y_1$ and $Y_2$, we can introduce a variable $X'$ to account for the effect of $Y_1$ and $X''$ for $Y_2$, and the factor $\phi(Y_1, Y_2, X)$ can be expressed as

$$\phi(y_1, y_2, x) = \sum_{x' \vee x'' = x} \psi(y_1, x')\gamma(y_2, x'') \tag{1}$$

where the summation is over all values $x'$ and $x''$ of $X'$ and $X''$ whose disjunction is equal to $x$. The $X$ variable is called *convergent* as it is where independent contributions from different sources are collected and combined. Non-convergent variables will be called *regular variables*.

Representing factors such as $\phi$ with $\psi$ and $\gamma$ is advantageous when the number of parents grows large, as the combined size of the component factors grows linearly, instead of exponentially.

Unfortunately, a straightforward use of VE for inference would lead to construct $O(2^n)$ tables where $n$ is the number of parents and the sum (1) will have an exponential number of terms. A modified algorithm, called VE1 [24], combines factors through a new operator $\otimes$:

$$\begin{aligned}\phi \otimes \psi(E_1 = \alpha_1, \ldots, E_k = \alpha_k, \boldsymbol{A}, \boldsymbol{B}_1, \boldsymbol{B}_2) = \\ \sum_{\alpha_{11} \vee \alpha_{12} = \alpha_1} \cdots \sum_{\alpha_{k1} \vee \alpha_{k2} = \alpha_k} \\ \phi(E_1 = \alpha_{11}, \ldots, E_k = \alpha_{k1}, \boldsymbol{A}, \boldsymbol{B}_1)\psi(E_1 = \alpha_{12}, \ldots, E_k = \alpha_{k2}, \boldsymbol{A}, \boldsymbol{B}_2)\end{aligned} \tag{2}$$

Here, $\phi$ and $\psi$ are two factors that share convergent variables $E_1 \ldots E_k$, $\boldsymbol{A}$ is the list of regular variables that appear in both $\phi$ and $\psi$ while $\boldsymbol{B}_1$ and $\boldsymbol{B}_2$ are the lists of variables appearing only in $\phi$ and $\psi$ respectively. By using the $\otimes$ operator, factors encoding the effect of parents can be combined in pairs, without the need to apply (1) on all factors at once.

Factors containing convergent variables are called *heterogeneous* while the remaining factors are called *homogeneous*. Heterogeneous factors sharing convergent variables must be combined with the operator $\otimes$, called *heterogeneous multiplication*.

Algorithm VE1 exploits causal independence by keeping two lists of factors: a list of homogeneous factors $\mathcal{F}_1$ and a list of heterogeneous factors $\mathcal{F}_2$. Procedure SUM-OUT is replaced by SUM-OUT1 that takes as input $\mathcal{F}_1$ and $\mathcal{F}_2$ and a variable $Z$ to be eliminated. First, all the factors containing $Z$ are removed from $\mathcal{F}_1$ and combined with multiplication to obtain factor $\phi$. Then all the factors containing $Z$ are removed from $\mathcal{F}_2$ and combined with heterogeneous multiplication obtaining $\psi$. If there are no such factors $\psi = nil$. In the latter case, SUM-OUT1 adds the new (homogeneous) factor $\sum_z \phi$ to $\mathcal{F}_1$, otherwise it adds the new (heterogeneous) factor $\sum_z \phi\psi$ to $\mathcal{F}_2$. Procedure VE1 is the same as VE with SUM-OUT replaced by SUM-OUT1 and with the difference that two sets of factors are maintained instead of one.

However VE1 is not correct for any elimination order. Correctness can be ensured by *deputising* the convergent variables: every such variable $E$ is replaced

by a new convergent variable $E'$ (called a *deputy variable*) in the heterogeneous factors containing it, so that $E$ becomes a regular variable. Finally, a new factor $\iota(E, E')$ is introduced, called *deputy factor*, that represents the identity function between $E$ and $E'$, i.e., it is defined by

| $\iota(E, E')$ | *ff* | *ft* | *tf* | *tt* |
|---|---|---|---|---|
| | 1.0 | 0.0 | 0.0 | 1.0 |

Deputising ensures that VE1 is correct as long as the elimination order is such that $\rho(E') < \rho(E)$.

### *2.5. GC-FOVE*

Work on lifting VE started with FOVE [3] and led to the definition of C-FOVE [5]. C-FOVE was refined in GC-FOVE [21], which represents the state of the art. Then, Gomes and Costa [7] adapted GC-FOVE to PFL.

*First-order Variable Elimination* (FOVE) [3, 4] computes the marginal probability distribution for a query random variable by repeatedly applying operators that are lifted counterparts of VE's operators. Models are in the form of a set of parfactors that are essentially the same as in PFL.

GC-FOVE tries to eliminate all (non-query) PRVs in a particular order by applying the following operations:

1. *Lifted Sum-Out*, that excludes a PRV from a parfactor $\phi$ if the PRV only occurs in $\phi$;
2. *Lifted Multiplication*, that multiplies two aligned parfactors. Matching variables must be properly aligned and the new coefficients must be computed taking into account the number of groundings in the constraints $C$;
3. *Lifted Absorption*, that eliminates $n$ PRVs that have the same observed value. If these operations cannot be applied, a chosen parfactor must be *split* so that some of its PRVs match another parfactor.

If no lifted operation can be executed, GC-FOVE completely grounds the PRVs and parfactors and performs inference on the ground level.

GC-FOVE considers also PRVs with counting formulas, introduced in C-FOVE [5]. A counting formula takes advantage of symmetries existing in factors that are products of independent variables. It represents a factor of the form $\phi(P(\mathtt{x}_1), P(\mathtt{x}_2), \ldots, P(\mathtt{x}_n))$, where all PRVs have the same domain, as $\phi(\#_{\mathtt{X}}[P(\mathtt{X})])$. The factor implements a multinomial distribution, such that its values depend on the number of variables $n$ and the domain size. The lifted counted variable is named a PCRV. PCRVs may result from summing-out, when we obtain parfactors with a single PRV, or through *Counting Conversion* that searches for factors of the form $\phi(\prod_i (S(\mathtt{X}_j) P(\mathtt{X}_j, \mathtt{Y}_i)))$ and counts on the occurrences of $\mathtt{Y}_i$.

GC-FOVE employs a constraint-tree to represent arbitrary constraints $\mathcal{C}$, whereas the PFL simply uses sets of tuples. Arbitrary constraints can capture more symmetries in the data, which potentially offers the ability to perform more operations at a lifted level.

## 3. LP$^2$

LP$^2$ [17] is an algorithm for performing lifted inference in ProbLog that translates the program into PFL and uses an extended GC-FOVE version for managing noisy-OR nodes.

### *3.1. Translating ProbLog into PFL*

In order to translate ProbLog into PFL, the program must be *tight*, i.e. must not contain positive cycles (see Section 6). If this condition if fulfilled, the ProbLog program can be converted first into a Bayesian network with noisy-OR nodes. Here we adapt the conversion for Logic Programs with Annotated Disjunctions presented in [12, 25] to the case of ProbLog.

The first step is to generate the grounding of the ProbLog program. For each atom $A$ in the Herbrand base of the program, the Bayesian network contains a Boolean random variable with the same name. Each probabilistic fact $p :: A$ is represented by a parentless node with the conditional probability table (CPT):

| $A$ | $f$ | $t$ |
|---|---|---|
| | 1-p | p |

For each ground rule $R_i = H \leftarrow B_1, \ldots, B_n, not(C_1), \ldots, not(C_m)$ we add to the network a random variable called $H_i$ that has as parents the random variables representing the atoms $B_1, \ldots, B_n, C_1, \ldots, C_m$ and the following CPT:

| $H_i$ | $B_1 = t, \ldots, B_n = t, C_1 = f, \ldots, C_m = f$ | all other columns |
|---|---|---|
| f | 0.0 | 1.0 |
| t | 1.0 | 0.0 |

In practice $H_i$ is the result of the conjunction of the random variables representing the atoms in the body. Then for each ground atom $H$ in the Herbrand base not appearing in a probabilistic fact, we add it to the network, with all $H_i$ in the ground rules with $H$ in the head as parents and with CPT:

| $H$ | at least one $H_i = t$ | all other columns |
|---|---|---|
| $f$ | 0.0 | 1.0 |
| $t$ | 1.0 | 0.0 |

representing the result of the disjunction of the random variables $H_i$. These families of random variables can be directly represented in PFL without the need to first ground the program, thus staying at the lifted level.

**Example 3 (Translation of a ProbLog program into PFL).** *The translation of the ProbLog program of Example 1 into PFL is*

```
bayes series1, self; [1, 0, 0, 1] ; [].
bayes series2, attends(P); [1, 0, 0, 1]; [person(P)].
bayes series, series1, series2 ; [1, 0, 0, 0, 0, 1, 1, 1]; [].
bayes attends1(P), at(P,A); [1, 0, 0, 1]; [person(P),attribute(A)].
```

```
bayes attends(P), attends1(P); [1, 0, 0, 1]; [person(P)].
bayes self; [0.9, 0.1]; [].
bayes at(P,A); [0.7, 0.3] ; [person(P),attribute(A)].
```

Notice that `series2` and `attends1(P)` can be seen as or-nodes, since they are in fact convergent variables. Thus, after grounding, factors derived from the second and the fourth parfactor should not be multiplied together but should be combined with heterogeneous multiplication.

To do so, we need to identify heterogeneous factors and add deputy variables and parfactors. We thus introduce two new types of parfactors to PFL, `het` and `deputy`. As mentioned before the type of a parfactor refers to the type of the network over which that parfactor is defined. These two new types are used in order to define a noisy-OR (Bayesian) network. The first parfactor is such that its ground instantiations are heterogeneous factors. The convergent variables are assumed to be represented by the first atom in the parfactor list of atoms. Lifting identity is straightforward, it corresponds to two atoms with an identity factor between their ground instantiations. Since the factor is fixed, it is not indicated.

**Example 4 (ProbLog program to PFL - LP$^2$).** *The translation of the Problog program of Example 1, shown in Example 3, is modified with the two new factors* `het` *and* `deputy` *as shown below:*

```
bayes series1p, self; [1, 0, 0, 1] ; [].
het series2p, attends(P); [1, 0, 0, 1]; [person(P)].
deputy series2, series2p; [].
deputy series1, series1p; [].
bayes series, series1, series2; [1, 0, 0, 0, 0, 1, 1, 1] ; [].
het attends1p(P), at(P,A); [1, 0, 0, 1]; [person(P),attribute(A)].
deputy attends1(P), attends1p(P); [person(P)].
bayes attends(P), attends1(P); [1, 0, 0, 1]; [person(P)].
bayes self; [0.9, 0.1]; [].
bayes at(P,A); [0.7, 0.3] ; [person(P),attribute(A)].
```

*Here,* `series1p`*,* `series2p` *and* `attends1p(P)` *are the convergent deputy random variables, and* `series1`*,* `series2` *and* `attends1(P)` *are their corresponding new regular variables. The fifth factor represents the OR combination of* `series1` *and* `series2` *to variable* `series`*.*

GC-FOVE must be modified in order to take into account heterogeneous factors and convergent variables. The VE algorithm must be replaced by VE1, i.e., two lists of factors must be maintained, one with homogeneous and the other with heterogeneous factors. When eliminating variables, homogeneous factors have higher priority and are combined with homogeneous factors only. Then heterogeneous factors are taken into account and combined before starting to mix factors from both types, to produce a final factor from which the selected random variable is eliminated.

**Operator** HET-MULTIPLY
**Inputs**:
(1) $\mathsf{G}_1 = \phi_1(\boldsymbol{A}_1)|\mathcal{C}_1$: a parfactor in model $G$ with convergent variables $\boldsymbol{A}_1 = \{A_{11}, \ldots, A_{1k}\}$
(2) $\mathsf{G}_2 = \phi_2(\boldsymbol{A}_2)|\mathcal{C}_2$: a parfactor in model $G$ with convergent variables $\boldsymbol{A}_2 = \{A_{21}, \ldots, A_{2k}\}$
(3) $\theta = \{\mathsf{X}_1 \to \mathsf{X}_2\}$: an alignment between $\mathsf{G}_1$ and $\mathsf{G}_2$
**Preconditions**:
(1) for $i = 1, 2$: $\mathsf{Y}_i = logvar(\boldsymbol{A}_i) \setminus \mathsf{X}_i$ is count-normalized w.r.t. $\mathsf{X}_i$ in $\mathcal{C}_i$
**Output**: $\phi(\boldsymbol{A})|\mathcal{C}$, such that
(1) $\mathcal{C} = \mathcal{C}_1\theta \bowtie \mathcal{C}_2$
(2) $\boldsymbol{A} = \boldsymbol{A}_1\theta \cup \boldsymbol{A}_2$
(3) Let $\boldsymbol{A}$ be $(A_1, \ldots, A_k, \boldsymbol{B})$ with $A_j = A_{1j}\theta = A_{2j}$ for $j = 1, \ldots, k$, $\boldsymbol{B}$ the set of regular variables
(4) for each assignment $\boldsymbol{a} = (a_1, \ldots, a_k, \boldsymbol{b})$ to $\boldsymbol{A}$ with $\boldsymbol{b}_1 = \pi_{\boldsymbol{A}_1\theta}(\boldsymbol{b})$, $\boldsymbol{b}_2 = \pi_{\boldsymbol{A}_2}(\boldsymbol{b})$
$\phi(a_1, \ldots, a_k, \boldsymbol{b}) =$
$\sum_{a_{11} \vee a_{21} = a_1} \cdots \sum_{a_{1k} \vee a_{2k} = a_k} \phi_1(a_{11}, \ldots, a_{1k}, \boldsymbol{b}_1)^{1/r_2} \phi_2(a_{21}, \ldots, a_{2k}, \boldsymbol{b}_2)^{1/r_1}$
with $r_i = \textsc{Count}_{\mathbf{Y}_i|\mathbf{X}_i}(\mathcal{C}_i)$
**Postcondition**: $G \sim G \setminus \{\mathsf{G}_1, \mathsf{G}_2\} \cup \{\text{HET-MULTIPLY}(\mathsf{G}_1, \mathsf{G}_2, \theta)\}$

Operator 1: Operator HET-MULTIPLY. Function COUNT is defined in [21].

Lifted heterogeneous multiplication (see Operator 1) considers the case in which the two factors share convergent random variables. PRVs must be *count-normalized*, i.e., the corresponding parameters must be scaled to take into account domain size and number of occurrences in the parfactor. This involves the use of the function COUNT: $\textsc{Count}_{\mathbf{Y}|\mathbf{X}}(\mathbf{x})$ returns the number of values for $\mathbf{Y}$ that co-occur with value $\mathbf{x}$, where $\mathbf{x} \in \mathcal{C}$. If this number is the same for each $\mathbf{x} \in \mathcal{C}$ then we can write $\textsc{Count}_{\mathbf{Y}|\mathbf{X}}(\mathcal{C})$ and we can say that $\mathbf{Y}$ is *count-normalized* w.r.t. $\mathbf{X}$ in $\mathcal{C}$. PRVs are then aligned and the joint domain is computed as the natural join between the set of constraints. Following standard lifted multiplication, we assume the same PRV will have a different instance in each grounded factor. We thus proceed as in the ground case, and, for each case $(a_{11}, \ldots, a_{1k}, \boldsymbol{b}_1, \boldsymbol{b}_2)$, we sum the potentials obtained by multiplying $\phi_1(a_{11}, \ldots, a_{1k}, \boldsymbol{b}_1)$ and $\phi_2(a_{11}, \ldots, a_{1k}, \boldsymbol{b}_2)$.

**Example 5.** *Consider the heterogeneous parfactors* $\mathsf{G}_1 = \phi_1(P(\mathtt{X}_1))|\mathcal{C}_1$ *and* $\mathsf{G}_2 = \phi_2(P(\mathtt{X}_2), Q(\mathtt{X}_2, \mathtt{Y}_2))|\mathcal{C}_2$ *and suppose that we want to multiply* $\mathsf{G}_1$ *and* $\mathsf{G}_2$*;* $P(\mathtt{X})$ *is convergent in* $\mathsf{G}_1$ *and* $\mathsf{G}_2$*;* $\{\mathtt{X}_1 \to \mathtt{X}_2\}$ *is an alignment between* $\mathsf{G}_1$ *and* $\mathsf{G}_2$*;* $\mathtt{Y}_2$ *is count-normalized w.r.t.* $\mathtt{X}_2$ *in* $\mathcal{C}_2$*;* $r_1 = \textsc{Count}_{\mathtt{Y}_1|\mathtt{X}_1}(\mathcal{C}_1) = 2$ *and* $\textsc{Count}_{\mathtt{Y}_2|\mathtt{X}_2}(\mathcal{C}_2) = 3$.

*Then* HET-MULTIPLY$(\mathsf{G}_1, \mathsf{G}_2, \{\mathtt{X}_1 \to \mathtt{X}_2\}) = \phi(P(\mathtt{X}_2), Q(\mathtt{X}_2, \mathtt{Y}_2))|\mathcal{C}$ *with* $\phi$ *given by:*

| | $\phi(P(\mathtt{X}_2), Q(\mathtt{X}_2, \mathtt{Y}_2))$ |
|---|---|
| *ff* | $\phi_1(f)^{1/3}\phi_2(f,f)^{1/2}$ |
| *ft* | $\phi_1(f)^{1/3}\phi_2(f,t)^{1/2}$ |
| *tf* | $\phi_1(f)^{1/3}\phi_2(t,f)^{1/2} + \phi_1(t)^{1/3}\phi_2(f,f)^{1/2} + \phi_1(t)^{1/3}\phi_2(t,f)^{1/2}$ |
| *tt* | $\phi_1(f)^{1/3}\phi_2(t,t)^{1/2} + \phi_1(t)^{1/3}\phi_2(f,t)^{1/2} + \phi_1(t)^{1/3}\phi_2(t,t)^{1/2}$ |

**Operator** HET-SUM-OUT
**Inputs**:
(1) $\mathsf{G} = \phi(\boldsymbol{A})|\mathcal{C}$: a parfactor in model $G$
(2) let $\boldsymbol{A} = (A_1, \ldots, A_k, A_{k+1}, \boldsymbol{B})$ where $A_1, \ldots, A_k$ are convergent variables
(3) $A_{k+1}$ is the variable to be summed out
**Preconditions**:
(1) For all PRVs $\mathcal{V}$, other than $A_{k+1}|\mathcal{C}$, in $G$: $RV(\mathcal{V}) \cap RV(A_{k+1}|\mathcal{C}) = \emptyset$
(2) $A_{k+1}$ contains all the logvars $\mathtt{X} \in logvar(\boldsymbol{A})$ for which $\pi_{\mathtt{X}}(\mathcal{C})$ is not a singleton
(3) $\mathbf{X}^{excl} = logvar(A_{k+1}) \setminus logvar(\boldsymbol{A} \setminus A_{k+1})$ is count-normalized w.r.t.
$\mathbf{X}^{com} = logvar(A_{k+1}) \cap logvar(\boldsymbol{A} \setminus A_{k+1})$ in $\mathcal{C}$
**Output**: $\phi'(\boldsymbol{A}')|\mathcal{C}'$, such that
(1) $\boldsymbol{A}' = \boldsymbol{A} \setminus A_{k+1}$
(2) $\mathcal{C}' = \pi_{\mathbf{X}}(\mathcal{C})$
(3) for each assignment $(\boldsymbol{a}', \boldsymbol{b}) = (a'_1, \ldots, a'_k, \boldsymbol{b})$ to $\boldsymbol{A}'$
$\phi'(\boldsymbol{a}', \boldsymbol{b}) =$
$\Big(\sum_{\boldsymbol{a} \leq \boldsymbol{a}'} \sum_{a_{k+1} \in range(A_{k+1})} \textsc{Mul}(A_{k+1}, a_{k+1}) \phi(a_1, \ldots, a_k, a_{k+1}, \boldsymbol{b})\Big)^r -$
$- \sum_{\boldsymbol{a} < \boldsymbol{a}'} \phi'(a_1, \ldots, a_k, \boldsymbol{b})$ with
$r = \textsc{Count}_{\mathbf{X}^{excl}|\mathbf{X}^{com}}(\mathcal{C})$
**Postcondition**: $\mathcal{P}_{G \setminus \{\mathsf{G}\} \cup \{\text{HET-SUM-OUT}(\mathsf{G}, (A_1, \ldots, A_k), A_{k+1})\}} = \sum_{RV(A_{k+1})} \mathcal{P}_G$

Operator 2: Operator HET-SUM-OUT. The order $\leq$ between truth values is the obvious one and between tuples of truth values is the product order induced by $\leq$ between values, i.e., $(a_1, \ldots, a_k) \leq (a'_1, \ldots, a'_k)$ iff $a_i \leq a'_i$ for $i = 1, \ldots, k$ and $\mathbf{a} < \mathbf{a}'$ iff $\mathbf{a} \leq \mathbf{a}'$ and $\mathbf{a}' \not\leq \mathbf{a}$. Function MUL is defined in [21].

The SUM-OUT operator must be modified as well to account for the case that random variables must be summed out from a heterogeneous factor. Let us suppose we want to eliminate the PRV $Q(\mathtt{X}, \mathtt{Y})$ from the parfactor $\phi(P(\mathtt{X}), Q(\mathtt{X}, \mathtt{Y}))|\mathcal{C}$ with $\mathcal{C} = \{\mathtt{x}_1, \mathtt{x}_2\} \times \{\mathtt{y}_1, \mathtt{y}_2\}$. This parfactor stands for four ground factors of the form $\phi(P(\mathtt{x}_i), Q(\mathtt{x}_i, \mathtt{y}_j))$ for $i, j = 1, 2$ where $P(\mathtt{x}_i)$ is convergent. Given an individual $\mathtt{x}_i$, the two factors $\phi(P(\mathtt{x}_i), Q(\mathtt{x}_i, \mathtt{y}_1))$ and $\phi(P(\mathtt{x}_i), Q(\mathtt{x}_i, \mathtt{y}_2))$ share a convergent variable and cannot be multiplied together with regular multiplication. Therefore, in order to sum out $Q(\mathtt{X}, \mathtt{Y})$, heterogeneous multiplication must first be used to combine the two factors. To avoid generating first the ground factors, we have added to GC-FOVE HET-SUM-OUT (Operator 2) that performs the combination and the elimination of a random variable at the same time.

**Example 6.** *Consider the heterogeneous parfactor* $\mathsf{G} = \phi(R, P(\mathtt{X}), Q(\mathtt{X}, \mathtt{Y}))|\mathcal{C}$ *and suppose that we want to sum out* $Q(\mathtt{X}, \mathtt{Y})$*, that* $R$ *and* $P(\mathtt{X})$ *are convergent, that* $\mathtt{Y}$ *is count-normalized w.r.t.* $\mathtt{X}$ *and that* $\textsc{Count}_{\mathtt{Y}|\mathtt{X}}(\mathcal{C}) = 2$*. Then* HET-SUM-OUT$(G, (R, P(\mathtt{X})), Q(\mathtt{X}, \mathtt{Y})) = \phi'(R, P(\mathtt{X}))|\mathcal{C}'$ *with* $\phi'$ *given by*

| | $\phi'(R, P(\mathtt{X}))$ |
|---|---|
| *ff* | $(\phi(f,f,f) + \phi(f,f,t))^2$ |
| *ft* | $(\phi(f,t,f) + \phi(f,t,t) + \phi(f,f,f) + \phi(f,f,t))^2 - \phi'(f,f)$ |
| *tf* | $(\phi(t,f,f) + \phi(t,f,t) + \phi(f,f,f) + \phi(f,f,t))^2 - \phi'(f,f)$ |
| *tt* | $(\phi(t,t,f) + \phi(t,t,t) + \phi(t,f,f) + \phi(t,f,t) + \phi(f,t,f) + \phi(f,t,t) + \phi(f,f,f) + \phi(f,f,t))^2 -$ $\phi'(t,f) - \phi'(f,t) - \phi'(f,f)$ |

## 4. Lifted Inference with Aggregation Parfactors

Kisynski and Poole [19] proposed an approach based on *aggregation parfactors* instead of parfactors. Aggregation parfactors are very expressive and can represent different kind of causal independence models, where noisy-OR and noisy-MAX are special cases. They are of the form $\langle \mathcal{C}, P, C, F_P, \boxtimes, \mathcal{C}_A \rangle$, where $P$ and $C$ are parameterized random variables which share all the parameters except one - let's say $\mathtt{A}$ which is in $P$ but not in $C$ - and the range of $P$ is a subset of that of $C$; $\mathcal{C}$ and $\mathcal{C}_\mathtt{A}$ are a set of inequality constraints respectively not involving and involving $\mathtt{A}$; $F_P$ is a factor from the range of $P$ to real values and $\boxtimes$ is a commutative and associative deterministic binary operator over the range of $C$.

When $\boxtimes$ is the MAX operator, of which the OR operator is a special case, a total ordering $\prec$ on the range of $C$ can be defined. An aggregation parfactor can be replaced with two parfactors of the form $\langle \mathcal{C} \cup \mathcal{C}_\mathtt{A}, \{P, C'\}, F_C \rangle$ and $\langle \mathcal{C}, \{C, C'\}, F_\Delta \rangle$, where $C'$ is an auxiliary parameterized random variable that has the same parameterization and range as $C$. Let $\mathbf{v}$ be an assignment of values to random variables, then $F_C(\mathbf{v}(P), \mathbf{v}(C')) = F_P(\mathbf{v}(P))$ when $\mathbf{v}(P) \preceq \mathbf{v}(C')$, 0 otherwise, while $F_\Delta(\mathbf{v}(C), \mathbf{v}(C')) = 1$ if $\mathbf{v}(C) = \mathbf{v}(C')$, $-1$ if $\mathbf{v}(C)$ is equal to a successor of $\mathbf{v}(C')$ and 0 otherwise.

When reasoning with ProbLog, we can use aggregation parfactors to model the dependency between the head of a rule and the body, when the body contains a single literal with an extra variable. In this case, given a grounding of the head, the contribution of all the ground clauses with that head must be combined by means of an OR. Since aggregation parfactors are replaced by regular parfactors, the technique can be used to reason with ProbLog by converting the program into PFL with these additional parfactors. The conversion is possible only if the ProbLog program is tight, i.e. does not contain positive cycles (see Section 6).

In the case of ProbLog the range of PRVs is binary and $\boxtimes$ is OR. For example, the clause `series2:- attends(P)` can be represented with the aggregation parfactor $\langle \emptyset, \mathtt{attends(P)}, \mathtt{series2}, F_P, \vee, \emptyset \rangle$, where $F_P(0) = 0$ and $F_P(1) = 1$. This is replaced by the parfactors $\langle \emptyset, \{\mathtt{attends(P)}, \mathtt{series2p}\}, F_C \rangle$, $\langle \emptyset, \{\mathtt{series2}, \mathtt{series2p}\}, F_\Delta \rangle$ with $F_C(0,0) = 1$, $F_C(0,1) = 1$, $F_C(1,0) = 0$, $F_C(1,1) = 1$, $F_\Delta(0,0) = 1$, $F_\Delta(0,1) = 0$, $F_\Delta(1,0) = -1$ and $F_\Delta(1,1) = 1$.

When the body of a rule contains more than one literal and/or more than one extra variable with respect to the head, the rule must be first split into multiple rules (adding auxiliary predicate names) satisfying the constraint.

**Example 7 (ProbLog program to PFL - aggregation parfactors).** *The program of Example 1 using the above encoding for aggregation parfactors is*

```
bayes series1p, self; [1, 0, 0, 1] ; [].
bayes series2p, attends(P) [1, 0, 1, 1]; [person(P)].
bayes series2, series2p; [1, 0, -1, 1]; [].
bayes series1, series1p; [1, 0, -1, 1]; [].
bayes series, series1, series2; [1, 0, 0, 0, 0, 1, 1, 1] ; [].
bayes attends1p(P), at(P,A); [1, 0, 1, 1];
```

```
  [person(P),attribute(A)].
bayes attends1(P), attends1p(P); [1, 0, -1, 1]; [person(P)].
bayes attends(P), attends1(P); [1, 0, 0, 1]; [person(P)].
bayes self; [0.9, 0.1]; [].
bayes at(P,A); [0.7, 0.3] ; [person(P),attribute(A)].
```

Thus, by using the technique of [19], we can perform lifted inference in ProbLog by a simple conversion to PFL, without the need to modify PFL algorithms.

## 5. Weighted First Order Model Counting

A different approach to lifted inference for PLPDS concerns the use of Weighted First Order Model Counting (WFOMC). WFOMC takes as input a triple $(\Delta, \mathrm{w}, \overline{\mathrm{w}})$, where $\Delta$ is a sentence in First Order Logic and w and $\overline{\mathrm{w}}$ are weight functions which associate a real number to positive and negative literals respectively. Given a triple $(\Delta, \mathrm{w}, \overline{\mathrm{w}})$ and a query $\phi$, its probability $P(\phi)$ is given by

$$P(\phi) = \frac{WFOMC(\Delta \wedge \phi, \mathrm{w}, \overline{\mathrm{w}})}{WFOMC(\Delta, \mathrm{w}, \overline{\mathrm{w}})}$$

Here, $WFOMC(\Delta, \mathrm{w}, \overline{\mathrm{w}})$ corresponds to the sum of the weights of all Herbrand models of $\Delta$, where the weight of a model is the product of its literal weights. Hence $WFOMC(\Delta, \mathrm{w}, \overline{\mathrm{w}}) = \sum_{\omega \models \Delta} \prod_{L \in \omega_0} \overline{\mathrm{w}}(pred(L)) \prod_{L \in \omega_1} \mathrm{w}(pred(L))$ where $\omega_0$ and $\omega_1$ are respectively false and true literals in the interpretation $\omega$ and *pred* maps literals $L$ to their predicate. Two lifted algorithms exist for exact WFOMC, one based on first-order knowledge compilation [6, 26, 27], and the other based on first-order DPLL search [28]. They both require the input theory to be in *first-order CNF*. A first-order CNF is a theory consisting of a conjunction of sentences of the form $\forall \mathtt{X}_1, ..., \forall \mathtt{X}_n, L_1 \vee ... \vee L_m$.

To encode a ProbLog program in a first-order CNF, Clark's completion [29] must be applied. For tight logic programs [30] Clark's completion is correct, in the sense that every model of the logic program is a model of the completion, and vice versa. The result is a set of rules in which each predicate is encoded by a single sentence. ProbLog rules are of the form $P(\mathtt{X}) :\!- B_i(\mathtt{X}, \mathtt{Y}_i)$ where $\mathtt{Y}_i$ is a variable that appears in the body $B_i$ but not in the head $P(\mathtt{X})$. The corresponding sentence in the completion is $\forall \mathtt{X}, P(\mathtt{X}) \Leftrightarrow \bigvee_i \exists \mathtt{Y}_i, B_i(\mathtt{X}, \mathtt{Y}_i)$. The program must be acyclic in order to make the completion sound, thus it is necessary to first remove positive loops [31].

Since WFOMC requires an input where existential quantifiers are absent, in [18] the authors presented a sound and modular Skolemization procedure to translate ProbLog programs into first-order CNF. Regular Skolemization cannot be used because it introduces function symbols, that are problematic for model counters. Therefore, existential quantifiers in expressions of the form $\exists \mathtt{X}, \phi(\mathtt{X}, \mathtt{Y})$ are replaced by the following formulas [18]:

$$\forall \mathtt{X}. Z(\mathtt{X})$$

$$\forall \mathbf{Y}, \forall \mathtt{X}, Z(\mathbf{Y}) \vee \neg\phi(\mathtt{X}, \mathbf{Y})$$

$$\forall \mathbf{Y}, S(\mathbf{Y}) \vee Z(\mathbf{Y})$$

$$\forall \mathbf{Y}, \forall \mathtt{X}, S(\mathbf{Y}) \vee \neg\phi(\mathtt{X}, \mathbf{Y})$$

Here $Z$ is the Tseitin predicate ($\mathrm{w}(Z) = \overline{\mathrm{w}}(Z) = 1$) and $S$ is the Skolem predicate ($\mathrm{w}(S) = 1$, $\overline{\mathrm{w}}(S) = -1$). This substitution can be used also for eliminating universal quantifiers since $\forall \mathtt{X}, \phi(\mathtt{X}, \mathbf{Y})$ can be seen as $\neg\exists \mathtt{X}, \neg\phi(\mathtt{X}, \mathbf{Y})$. Once no more substitutions can be applied, the resulting program can be turned into first-order CNF with standard transformations.

This replacement introduces a relaxation of the theory, thus more models are found. However, for every additional model with weight $W$, there is exactly one additional model with weight $-W$, thus the WFOMC does not change. The interaction between the three relaxed formulas, the equivalence between $Z(\mathbf{Y})$ and $\exists \mathtt{X}, \phi(\mathtt{X}, \mathbf{Y})$, and the model weights follow the behaviour:

1. when $Z(\mathbf{Y})$ is false, then $\exists \mathtt{X}, \phi(\mathtt{X}, \mathbf{Y})$ is false while $S(\mathbf{Y})$ is true and the weight is 1.
2. when $Z(\mathbf{Y})$ is true, then either: (a) $\exists \mathtt{X}, \phi(\mathtt{X}, \mathbf{Y})$ is true and $S(\mathbf{Y})$ is true, or (b) $\exists \mathtt{X}, \phi(\mathtt{X}, \mathbf{Y})$ is false and $S(\mathbf{Y})$ is true, in which case the state has a positive weight $W$, or (c) $\exists \mathtt{X}, \phi(\mathtt{X}, \mathbf{Y})$ is true and $S(\mathbf{Y})$ is false, in which case the state has weight $-W$. It is easy to note that the last two cases cancel out.

The WFOMC encoding for a ProbLog program exploits two mapping functions which associate the probability $p$ and $1 - p$ of a probabilistic fact with the true and false values of the predicate respectively. After the application of Clark's completion, the result may not be in Skolem normal form thus the techniques described above must be applied before executing WFOMC. The system WFOMC[2] solves the WFOMC problem by compiling the input theory into First Order d-DNNF diagrams [32, 33].

**Example 8 (ProbLog program to Skolem normal form).** *The translation of the ProbLog program of Example 1 into the WMC input format of the WFOMC system is*

```
predicate series1 1 1
predicate series2 1 1
predicate self 0.1 0.9
predicate at(P,A) 0.3 0.7
predicate z1 1 1
predicate s1 1 -1
predicate z2 1 1
predicate s2 1 -1
```

[2] https://dtai.cs.kuleuven.be/software/wfomc

```
series v ! z1
!series v z1
z1 v !self
z1 v !attends(P)
z1 v s1
s1 v !self
s1 v !attends(P)

attends(P) v ! z2(P)
!attends(P) v  z2(P)
z2(P) v !at(P,A)
z2(P) v s2(P)
s2(P) v !at(P,A)
```

*Here,* `predicate` *is the mapping function for the probability values while* `z1` *and* `z2` *are Tseitin predicates and* `s1` *and* `s2` *are Skolem predicates.*

## 6. Non-Tight Logic Programs

$LP^2$ and aggregation parfactors, described in Section 3 and 4 respectively, require a conversion from ProbLog to PFL for performing inference. The first step of this translation is the transformation of a ProbLog program into a Bayesian network with noisy-OR nodes. However, since Bayesian networks cannot have cycles, this conversion is not correct if the program is *non-tight* in the sense proposed by Fages [30][3], i.e., if the program contains positive cycles. A similar problem occurs with WFOMC: Clark's completion [29] is correct only for tight logic programs.

Fages in [30] proved that if an LP program is tight then the Herbrand models of its Clark's completion [29] are minimal and coincide with the stable models of the original LP program. The consequence of this theoretical result is that, if the ProbLog program is tight, we can correctly convert it into a first-order theory by means of Clark's completion.

To apply these techniques to non-tight programs, we need to remove positive loops. We could first apply a conversion using the method proposed in [31] that converts normal logic programs to *atomic normal programs* then to clauses. An atomic normal program contains only rules of the form

$$A \text{ :- } not\ C_1, \ldots, not\ C_m.$$

where $A$ and $C_i$ are atoms and *not* denotes Clark's *negation as failure to prove*. Such programs are tight and, as a consequence, it is possible to translate them into PFL programs and to use Clark's completion.

[3] In [30] the programs are not called *tight*, but *positive-order-consistent*.

However this conversion was proposed only for the case of ground LPs. Proposing a conversion for non-ground programs is an interesting direction for future work, especially if function symbols are allowed.

## 7. Experiments

In order to evaluate the performance of the three techniques presented in the paper, LP$^2$, C-FOVE with aggregation parfactors (C-FOVE-AP) and WFOMC, we applied them to five problems:

- *workshops attributes* [5]
- two different versions of *competing workshops* [5]
- two different versions of Example 7 in [34], that we call *plates*

All the tests were done on a machine with an Intel Dual Core E6550 2.33GHz processor and 4GB of main memory. For $LP^2$ we used the implementation available in the PFL package of Yap Prolog while for C-FOVE-AP we used the solver `lve` of the PFL package that implements C-FOVE. `lve` implements the algorithm GC-FOVE. For WFOMC we used version 3.0 of the WFMOC system.

The *workshop attributes* problem is defined as:

```
series:- person(P),attends(P),sa(P).
0.501::sa(P):-person(P).
attends(P):- person(P),attr(A),at(P,A).
0.3::at(P,A):-person(P),attr(A).
```

This problem differs from Example 1 because the first clause for `series` is missing and the second clause contains a probabilistic atom in its body, i.e., `sa`.

The *competing workshops* problems differ from *workshop attributes* because they model, instead of workshop attributes, a set of competing workshops `W` each one associated with a binary random variable `hot(W)`, which indicates whether it is focusing on popular research areas. In one set of experiments, called *competing workshops PH*, we associated a probability value to `hot(W)` while in the second, *competing workshops CH*, `hot(W)` is certain. The ProbLog program corresponding to *competing workshops PH* is the following.

```
series:- person(P),attends(P),sa(P).
0.501::sa(P):-person(P).
attends(P):- person(P),\+ attends_other(P).
attends_other(P):- person(P),workshop(W),hot(W),ah(P,W).
0.8::ah(P,W) :- person(P),workshop(W).
0.51::hot(W) :- workshop(W).
```

The code for *competing workshops CH* is obtained from the above by simply removing the probability annotation from the last clause.

The *plates* problem is an artificial example designed to challenge lifted inference systems. The author in [34] stated that solving it in time less than linear

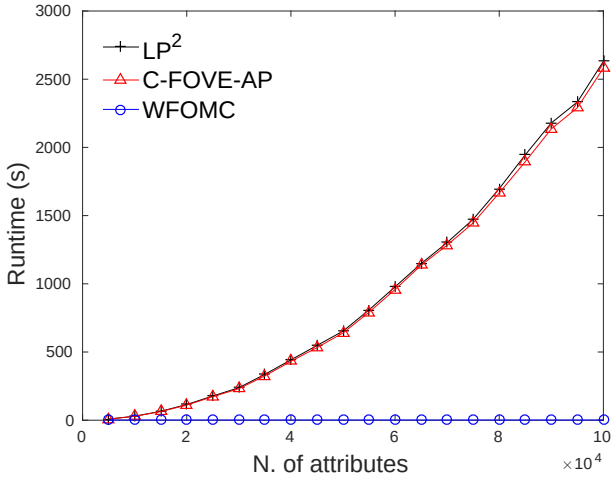


Figure 1: Runtime of LP$^2$, C-FOVE with aggregation parfactors and WFOMC for the *workshops attributes* problem.

in the population size is part of ongoing research. The problem contains two sets of individuals, $X$ and $Y$. The distribution is defined by 7 probabilistic facts and 9 rules. In particular, in one set of experiments, called *plates* $X$, we fixed the number of $Y$ individuals while we varied the number of $X$ individuals, in the second set of experiments (*plates* $Y$) we fixed the number of $X$ individuals and we varied the number of $Y$s. The ProbLog program corresponding to the *plates* problem without individuals is reported in Appendix A.3.
Figure 1 shows the runtime of LP$^2$, C-FOVE with aggregation parfactors (C-FOVE-AP) and WFOMC on the *workshop attributes* problem for the query `series`, where we fixed the number of people to 50 and we increased the number of attributes $m$.

Figure 2 and Figure 3 show the runtime of the three systems on the *competing workshops PH* and *competing workshops CH* problems respectively for the query `series`, with 10 competing workshops and an increasing number $n$ of people.

Figure 4 and Figure 5 show the runtime for *plates* $X$ and *plates* $Y$ respectively for the query `f` where we fixed the number of different $Y$ and $X$ individuals respectively to 5 and we increased the number of the individuals for the other variable. The appendix shows the encoding of these problems in PFL and WMC.

According to [35],[26], function-free first-order logic with equality and 2 variables per formula (2-`FFFOL`$(=)$) is domain-liftable, i.e., the complexity of reasoning is polynomial in the domain size. Since all these problems fall in 2-`FFFOL`$(=)$, we expect the algorithms to run in polynomial time. As the results show, all the systems can manage domains that are very large. Moreover, LP$^2$ and C-FOVE with aggregation parfactors show approximately the same performance on all

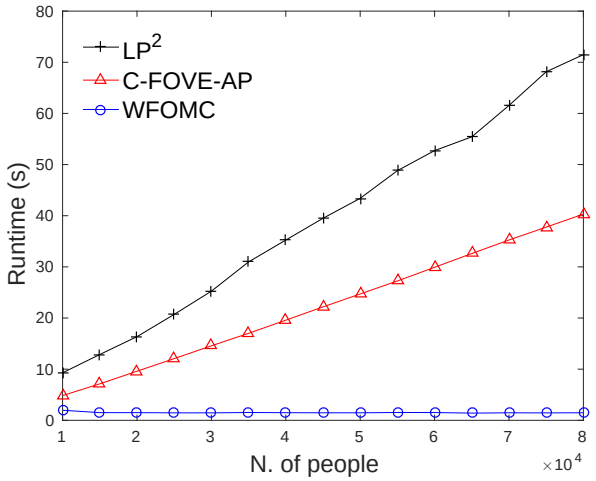


Figure 2: Runtime of LP$^2$, C-FOVE with aggregation parfactors and WFOMC for the *competing workshops PH* problem.

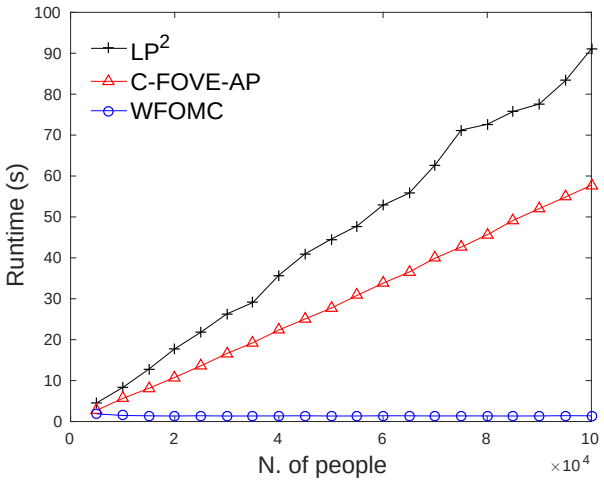


Figure 3: Runtime of LP$^2$, C-FOVE with aggregation parfactors and WFOMC for the *competing workshops CH* problem.

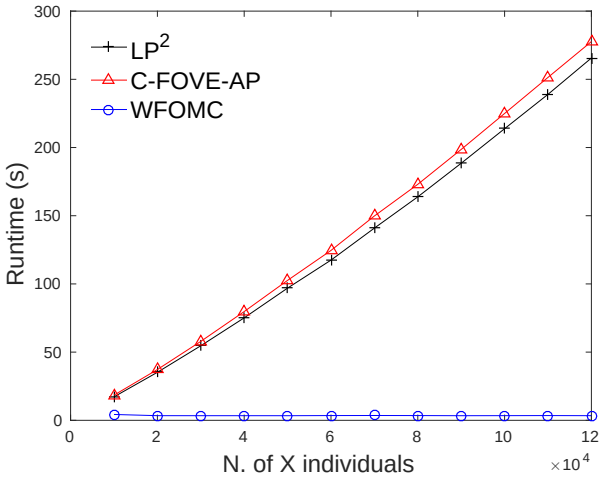


Figure 4: Runtime of LP$^2$, C-FOVE with aggregation parfactors and WFOMC for the *plates* $X$ problem.

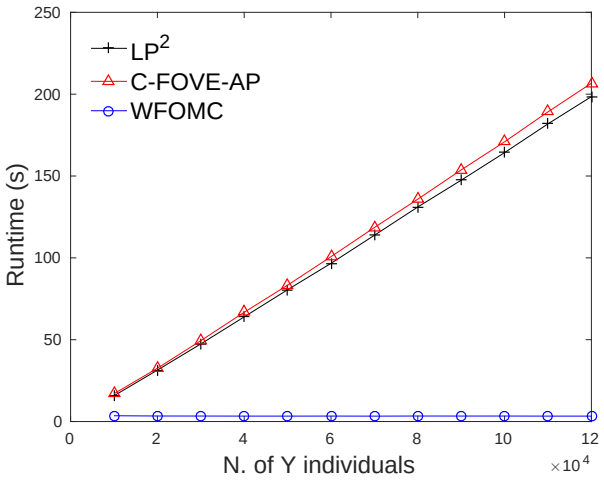


Figure 5: Runtime of LP$^2$, C-FOVE with aggregation parfactors and WFOMC for the *plates* $Y$ problem.

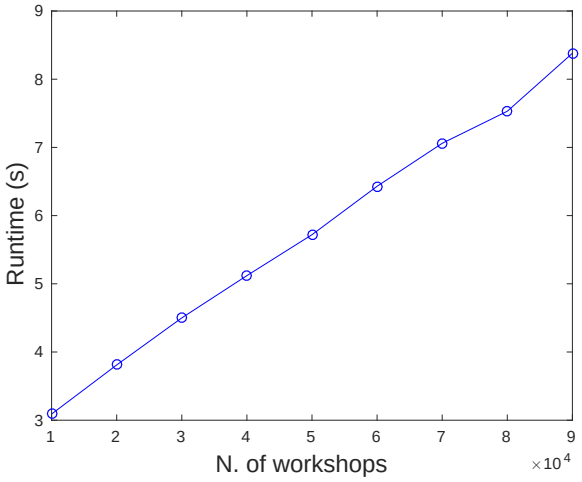


Figure 6: Runtime of WFOMC for the *competing workshops PH* problem. The number of people is fixed to 100,000 (setting 1).

problems, with running time increasing roughly linearly with the domain size, apart from the workshop attributes problem where it grows quadratically.

WFOMC uses exponentiation to the power of the domain size so the time must be logarithmic in the domain size [36], however the multiplicative constant factors involved are probably so small that they make the time appear constant.

To investigate more in depth the results for WFOMC, we tested it with larger domains on *competing workshops PH* and *plates*.

For the first problem we considered three different settings:

1. we fixed the number of people to 100,000 while increasing the number of workshops from 10,000 to 100,000;
2. we fixed the number of workshops to 10,000 while increasing the number of people from 1000 to 10,000;
3. we kept the number of people and workshops equal and we increased it from 100 to 10,000.

Results of the first two settings are shown in Figure 6 and Figure 7. In the first setting WFOMC runs in a time that is roughly linear with a small constant in the size of the domain. In both tests, with the largest size, i.e., 100,000 in the first setting and 10,000 in the second one, the process started thrashing. In particular, in the first setting the computation exceeded 24 hours, while in the second one the process was killed by the operating system.

Figure 8 shows the results of the third setting, where the $X$-axis shows the product of the number of workshops and people. Again the trend is linear, albeit with a larger constant.

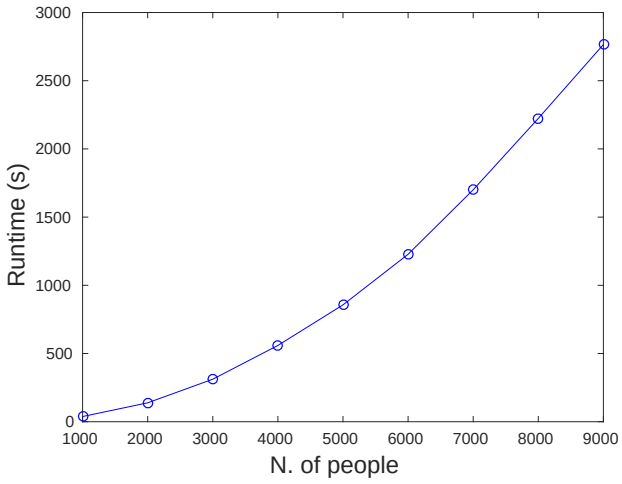


Figure 7: Runtime of WFOMC for the *competing workshops PH* problem. The number of workshops is fixed to 10,000 (setting 2).

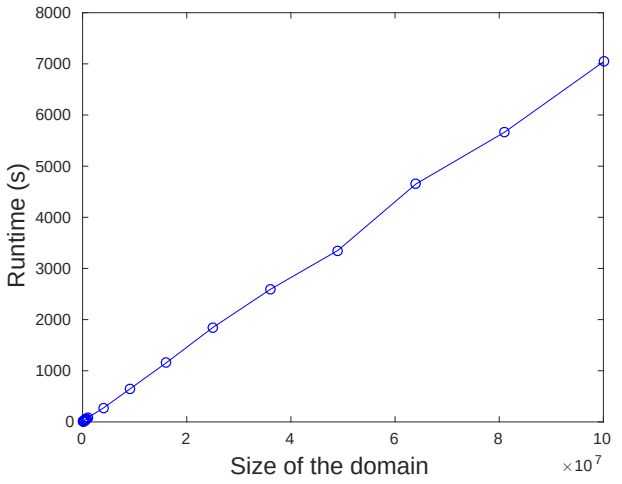


Figure 8: Runtime of WFOMC for the *competing workshops PH* problem. The number of both people and workshops increases (setting 3). The $X$-axis shows the product of the number of people and workshops.

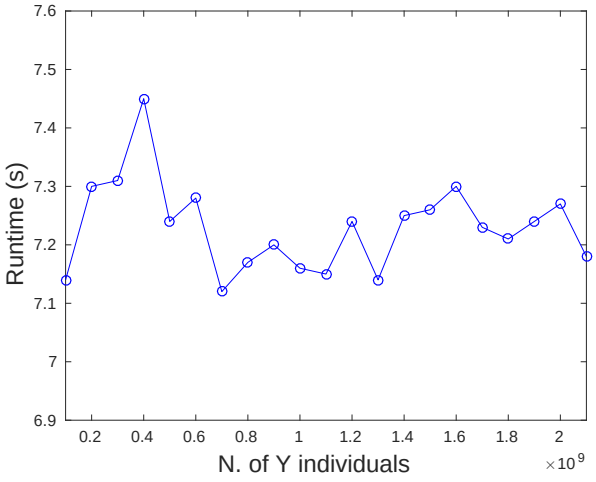


Figure 9: Runtime of WFOMC for the *plates* problem. The size of the domain of variable $X$ is fixed to 10,000.

Similarly to *competing workshops PH*, for *plates* we set up two different experiments:

1. we fixed the size of the domain of variable $X$ to 10,000 and we varied the domain size of variable $Y$ from $10^8$ to $2.1 \times 10^9$;
2. we kept the domain size of $X$ and $Y$ equal and we increased it from 1000 to $10^7$, so that the domain of the couple $(X, Y)$ ranges from $10^6$ to $10^{14}$.

Figure 9 and Figure 10 show the running time in seconds for the two settings. From these figures we can observe a roughly constant running time when the domain of only one variable is varied and a linear time when the domains of both variables are varied. In [34] Poole stated that the problem of performing inference in less than linear time in the size of the domains of the two variables was still open. These experiments show that when keeping one domain constant, the problem is solved, while it is not when the size of the combined domain is taken into account. In any case, WFOMC is the clear winner for performing lifted inference on probabilistic logic programs.

## 8. Conclusions

While this article does not aim at being a complete account of the activity in the field, we hope to have given an introduction that highlights the important results already achieved for lifted inference, supported by an experimental evaluation of different techniques.

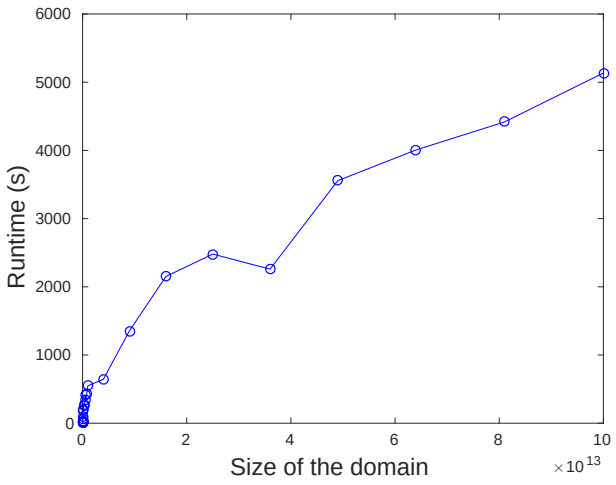


Figure 10: Runtime of WFOMC for the *plates* problem. The size of the domain of both variables $X$ and $Y$ increases. The $X$-axis is the size of the combined domain, the product of the domain size for the two variables.

The use of lifted inference can really speed up both inference and learning processes since it may exempt from taking into consideration all the individuals of the domain.

Among the proposed systems, $LP^2$ and aggregation parfactors use variable elimination, while WFOMC is based on knowledge compilation by means of First Order d-DNNF diagrams. The experimental results demonstrate that WFOMC is able to perform inference in a time that is linearly dependent on the domains of the variables in the considered models, while $LP^2$ and aggregation parfactors show a polynomial increase in the inference time.

## Acknowledgments

This work was supported by "National Group of Computing Science (GNCS-INDAM)". The authors would like to thank the anonymous reviewers of [17] for drawing their attention to [19] and [18].

[1] L. Getoor, B. Taskar (Eds.), Introduction to Statistical Relational Learning, MIT Press, 2007.

[2] L. De Raedt, P. Frasconi, K. Kersting, S. Muggleton (Eds.), Probabilistic Inductive Logic Programming - Theory and Applications, Vol. 4911 of LNCS, Springer, 2008.

[3] D. Poole, First-order probabilistic inference, in: G. Gottlob, T. Walsh (Eds.), 18th International Joint Conference on Artificial Intelligence, Morgan Kaufmann Publishers Inc., 2003, pp. 985–991.

[4] R. de Salvo Braz, E. Amir, D. Roth, Lifted first-order probabilistic inference, in: L. P. Kaelbling, A. Saffiotti (Eds.), 19th International Joint Conference on Artificial Intelligence, Professional Book Center, 2005, pp. 1319–1325.

[5] B. Milch, L. S. Zettlemoyer, K. Kersting, M. Haimes, L. P. Kaelbling, Lifted probabilistic inference with counting formulas, in: D. Fox, C. P. Gomes (Eds.), 23rd AAAI Conference on Artificial Intelligence, AAAI Press, 2008, pp. 1062–1068.

[6] G. Van den Broeck, N. Taghipour, W. Meert, J. Davis, L. D. Raedt, Lifted probabilistic inference by first-order knowledge compilation, in: T. Walsh (Ed.), 21st International Joint Conference on Artificial Intelligence, IJCAI/AAAI, 2011, pp. 2178–2185.

[7] T. Gomes, V. S. Costa, Evaluating inference algorithms for the prolog factor language, in: F. Riguzzi, F. Železný (Eds.), 22nd International Conference on Inductive Logic Programming, Vol. 7842 of LNCS, Springer, 2012, pp. 74–85.

[8] T. Sato, A statistical learning method for logic programs with distribution semantics, in: L. Sterling (Ed.), 12th International Conference on Logic Programming, MIT Press, 1995, pp. 715–729.

[9] D. Poole, Probabilistic horn abduction and Bayesian networks, Artif. Intell. 64 (1) (1993) 81–129.

[10] T. Sato, Y. Kameya, PRISM: a language for symbolic-statistical modeling, in: 15th International Joint Conference on Artificial Intelligence, Vol. 97, 1997, pp. 1330–1339.

[11] D. Poole, The Independent Choice Logic for modelling multiple agents under uncertainty, Artif. Intell. 94 (1997) 7–56.

[12] J. Vennekens, S. Verbaeten, M. Bruynooghe, Logic programs with annotated disjunctions, in: B. Demoen, V. Lifschitz (Eds.), 20th International Conference on Logic Programming, Vol. 3131 of LNCS, Springer, 2004, pp. 195–209.

[13] L. De Raedt, A. Kimmig, H. Toivonen, ProbLog: A probabilistic Prolog and its application in link discovery, in: M. M. Veloso (Ed.), 20th International Joint Conference on Artificial Intelligence, Vol. 7, AAAI Press, 2007, pp. 2462–2467.

[14] D. Fierens, G. V. den Broeck, J. Renkens, D. S. Shterionov, B. Gutmann, I. Thon, G. Janssens, L. De Raedt, Inference and learning in probabilistic logic programs using weighted boolean formulas, Theor. Pract. Log. Prog. 15 (3) (2015) 358–401.

[15] L. De Raedt, A. Kimmig, Probabilistic (logic) programming concepts, Mach. Learn. 100 (1) (2015) 5–47.

[16] A. Kimmig, L. Mihalkova, L. Getoor, Lifted graphical models: a survey, Mach. Learn. 99 (1) (2015) 1–45.

[17] E. Bellodi, E. Lamma, F. Riguzzi, V. S. Costa, R. Zese, Lifted variable elimination for probabilistic logic programming, Theor. Pract. Log. Prog. 14 (4-5) (2014) 681–695. doi:10.1017/S1471068414000283.

[18] G. Van den Broeck, W. Meert, A. Darwiche, Skolemization for weighted first-order model counting, in: C. Baral, G. D. Giacomo, T. Eiter (Eds.), 14th Fourteenth International Conference on Principles of Knowledge Representation and Reasoning, AAAI Press, 2014, pp. 111–120.

[19] J. Kisynski, D. Poole, Lifted aggregation in directed first-order probabilistic models, in: C. Boutilier (Ed.), 24th International Joint Conference on Artificial Intelligence, 2009, pp. 1922–1929.

[20] F. Riguzzi, The distribution semantics for normal programs with function symbols, Int. J. Approx. Reason. 77 (2016) 1 – 19. doi:10.1016/j.ijar.2016.05.005.

[21] N. Taghipour, D. Fierens, J. Davis, H. Blockeel, Lifted variable elimination: Decoupling the operators from the constraint language, J. Artif. Intell. Res. 47 (2013) 393–439.

[22] J. Kisynski, D. Poole, Constraint processing in lifted probabilistic inference, in: J. Bilmes, A. Y. Ng (Eds.), 25th Conference on Uncertainty in Artificial Intelligence, AUAI Press, 2009, pp. 293–302.

[23] N. L. Zhang, D. Poole, A simple approach to bayesian network computations, in: 10th Canadian Conference on Artificial Intelligence, 1994, pp. 171–178.

[24] N. L. Zhang, D. L. Poole, Exploiting causal independence in bayesian network inference, J. Artif. Intell. Res. 5 (1996) 301–328.

[25] W. Meert, J. Struyf, H. Blockeel, Learning ground CP-Logic theories by leveraging Bayesian network learning techniques, Fund. Inform. 89 (1) (2008) 131–160.

[26] G. Van den Broeck, On the completeness of first-order knowledge compilation for lifted probabilistic inference, in: J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. C. N. Pereira, K. Q. Weinberger (Eds.), 25th Annual Conference on Neural Information Processing Systems, 2011, pp. 1386–1394.

[27] G. Van den Broeck, Lifted inference and learning in statistical relational models, Ph.D. thesis, Ph. D. Dissertation, KU Leuven (2013).

[28] V. Gogate, P. M. Domingos, Probabilistic theorem proving, in: F. G. Cozman, A. Pfeffer (Eds.), 27th Conference on Uncertainty in Artificial Intelligence, AUAI Press, 2011, pp. 256–265.

[29] K. L. Clark, Negation as failure, in: Logic and Data Bases, 1977, pp. 293–322.

[30] F. Fages, Consistency of clark's completion and existence of stable models, Meth. of Logic in CS 1 (1) (1994) 51–60.

[31] T. Janhunen, Representing normal programs with clauses, in: R. L. de Mántaras, L. Saitta (Eds.), 16th Eureopean Conference on Artificial Intelligence including Prestigious Applicants of Intelligent Systems, IOS Press, 2004, pp. 358–362.

[32] A. Darwiche, A logical approach to factoring belief networks, in: D. Fensel, F. Giunchiglia, D. L. McGuinness, M. Williams (Eds.), 8th International Conference on Principles and Knowledge Representation and Reasoning, Morgan Kaufmann, 2002, pp. 409–420.

[33] M. Chavira, A. Darwiche, On probabilistic inference by weighted model counting, Artif. Intell. 172 (6-7) (2008) 772–799.

[34] D. Poole, The Independent Choice Logic and beyond, in: L. De Raedt, P. Frasconi, K. Kersting, S. Muggleton (Eds.), Probabilistic Inductive Logic Programming, Vol. 4911 of LNCS, Springer, 2008, pp. 222–243.

[35] M. Jaeger, G. Van den Broeck, Liftability of probabilistic inference: Upper and lower bounds, in: 2nd International Workshop on Statistical Relational AI, 2012, pp. 1–8.

[36] D. M. Gordon, A survey of fast exponentiation methods, J. Algorithms 27 (1) (1998) 129–146.

## Appendix A. Problems code

In this section the Problog and PFL code of the tested domains can be found.

### *Appendix A.1. Workshops Attributes*

To all programs of this section we added 50 workshops and an increasing number of attributes.

*PFL program for $LP^2$.*

```
het series1,ch1(P);[1.0, 0.0, 0.0, 1.0];[person(P)].

deputy series,series1;[].

bayes ch1(P),attends(P),sa(P);[1.0,1.0,1.0,0.0,
                               0.0,0.0,0.0,1.0];[person(P)].

bayes sa(P);[0.499,0.501];[person(P)].

het attends1(P),at(P,A);[1.0, 0.0, 0.0, 1.0];[person(P),attr(A)].

deputy attends(P),attends1(P);[person(P)].

bayes at(P,A);[0.7,0.3];[person(P),attr(A)].
```

*PFL program with Aggregation Parfactors.*

```
bayes series,series1;[1.0, 0.0, -1.0, 1.0];[].

bayes series1,ch1(P);[1.0, 0.0, 1.0, 1.0];[person(P)].

bayes ch1(P),attends(P),sa(P);[1.0,1.0,1.0,0.0,
                               0.0,0.0,0.0,1.0];[person(P)].

bayes sa(P);[0.499,0.501];[person(P)].

bayes attends1(P),at(P,A);[1.0, 0.0, 1.0, 1.0];[person(P),attr(A)].

bayes attends(P),attends1(P);[1.0, 0.0, -1.0, 1.0];[person(P)].

bayes at(P,A);[0.7,0.3];[person(P),attr(A)].
```

*WFOMC program.*

```
predicate series 1 1
predicate attends(P) 1 1
predicate sa(P) 0.501 0.499
predicate at(P,A) 0.3 0.7
predicate z1 1 1
predicate s1 1 -1
predicate z2(P) 1 1
predicate s2(P) 1 -1


series v !z1
!series v z1
z1 v  !attends(P) v !sa(P)
z1 v s1
s1 v  !attends(P) v !sa(P)

attends(P) v !z2(P)
!attends(P) v z2(P)
z2(P) v !at(P,A)
z2(P) v s2(P)
s2(P) v !at(P,A)
```

*Appendix A.2. Competing Workshops*

For the *competing workshops* problem we report only the PFL version. For testing purposes we added 10 workshops and an increasing number of people.

*PFL program for $LP^2$.*

```
bayes ch1(P),attends(P),sa(P);[1.0,1.0,1.0,0.0,
                               0.0,0.0,0.0,1.0];[person(P)].

het series1,ch1(P);[1.0, 0.0, 0.0, 1.0];[person(P)].

deputy series,series1;[].

bayes sa(P);[0.499,0.501];[person(P)].

het attends1(P),ch2(P,W);[1.0, 0.0, 0.0, 1.0];[person(P),workshop(W)].

deputy attends(P),attends1(P);[person(P)].

bayes ch2(P,W),hot(W),ah(P,W);[1.0,1.0,1.0,0.0,
                               0.0,0.0,0.0,1.0];[person(P),workshop(W)].

bayes ah(P,W);[0.2,0.8];[person(P),workshop(W)].
```

For *competing workshops PH*, the program contains also

```
bayes hot(W);[0.49,0.51];[workshop(W)].
```

*PFL program with Aggregation Parfactors.*

```
bayes ch1(P),attends(P),sa(P);[1.0,1.0,1.0,0.0,
                               0.0,0.0,0.0,1.0];[person(P)].

bayes series,series1;[1.0, 0.0, -1.0, 1.0];[].

bayes series1,ch1(P);[1.0, 0.0, 1.0, 1.0];[person(P)].

bayes sa(P);[0.499,0.501];[person(P)].

bayes attends1(P),ch2(P,W);[1.0, 0.0, 1.0, 1.0];[person(P),workshop(W)].

bayes attends(P),attends1(P);[1,0,-1,1];[person(P)].

bayes ch2(P,W),hot(W),ah(P,W);[1.0,1.0,1.0,0.0,
                               0.0,0.0,0.0,1.0];[person(P),workshop(W)].

bayes ah(P,W);[0.2,0.8];[person(P),workshop(W)].
```

For *competing workshops PH*, the program contains also

```
bayes hot(W);[0.49,0.51];[workshop(W)].
```

*WFOMC program.*

```
predicate series 1 1
predicate attends(P) 1 1
predicate sa(P) 0.501 0.499
predicate ah(P,W) 0.8 0.2
```

```
predicate hot(W) 1 1
predicate z1 1 1
predicate s1 1 -1
predicate z2(P) 1 1
predicate s2(P) 1 -1


series v !z1
!series v z1
z1 v  !attends(P) v !sa(P)
z1 v s1
s1 v  !attends(P) v !sa(P)

attends(P) v ! z2(P)
!attends(P) v  z2(P)
z2(P) v !ah(P,W) v!hot(W)
z2(P) v s2(P)
s2(P) v !ah(P,W)v!hot(W)
```

For *competing workshops PH*, the definition of predicate `hot` is

```
predicate hot(W) 0.51 0.49
```

*Appendix A.3. Plates*

For the *plates* $Y$ (*plates* $X$) problem we added 5 individuals for $X$ ($Y$) and an increasing number of individuals for $Y$ ($X$).

*ProbLog program.*

```
f:- e(Y).

e(Y) :- d(Y),n1(Y).
e(Y) :- y(Y),\+ d(Y),n2(Y).

d(Y):- c(X,Y).

c(X,Y):- b(X),n3(X,Y).
c(X,Y):- x(X),\+ b(X),n4(X,Y).

b(X):- a, n5(X).
b(X):- \+ a,n6(X).

a:- n7.

0.1::n1(Y) :-y(Y).
0.2::n2(Y) :-y(Y).
0.3::n3(X,Y) :- x(X),y(Y).
0.4::n4(X,Y) :- x(X),y(Y).
0.5::n5(X) :-x(X).
0.6::n6(X) :-x(X).
0.7::n7.
```

*PFL program for LP$^2$.*

```
het f1,e(Y);[1.0, 0.0, 0.0, 1.0];[y(Y)].

deputy f,f1;[].
```

```
bayes e1(Y),d(Y),n1(Y);[1.0, 1.0, 1.0, 0.0,
                        0.0, 0.0, 0.0, 1.0];[y(Y)].

bayes e2(Y),d(Y),n2(Y);[1.0, 0.0, 1.0, 1.0,
                        0.0, 1.0, 0.0, 0.0];[y(Y)].

bayes e(Y),e1(Y),e2(Y);[1.0, 0.0, 0.0, 0.0,
                        0.0, 1.0, 1.0, 1.0];[y(Y)].

het d1(Y),c(X,Y);[1.0, 0.0, 0.0, 1.0];[x(X),y(Y)].

deputy d(Y),d1(Y);[y(Y)].

bayes c1(X,Y),b(X),n3(X,Y);[1.0, 1.0, 1.0, 0.0,
                            0.0, 0.0, 0.0, 1.0];[x(X),y(Y)].

bayes c2(X,Y),b(X),n4(X,Y);[1.0, 0.0, 1.0, 1.0,
                            0.0, 1.0, 0.0, 0.0];[x(X),y(Y)].

bayes c(X,Y),c1(X,Y),c2(X,Y);[1.0, 0.0, 0.0, 0.0,
                              0.0, 1.0, 1.0, 1.0];[x(X),y(Y)].

bayes b1(X),a,n5(X);[1.0, 1.0, 1.0, 0.0,
                     0.0, 0.0, 0.0, 1.0];[x(X)].

bayes b2(X),a,n6(X);[1.0, 0.0, 1.0, 1.0,
                     0.0, 1.0, 0.0, 0.0];[x(X)].

bayes b(X),b1(X),b2(X);[1.0, 0.0, 0.0, 0.0,
                        0.0, 1.0, 1.0, 1.0];[x(X)].

bayes a,n7;[1.0, 0.0, 0.0, 1.0];[].

bayes n1(Y);[0.9, 0.1];[y(Y)].
bayes n2(Y);[0.8, 0.2];[y(Y)].
bayes n3(X,Y);[0.7, 0.3];[x(X),y(Y)].
bayes n4(X,Y);[0.6, 0.4];[x(X),y(Y)].
bayes n5(X);[0.5, 0.5];[x(X)].
bayes n6(X);[0.4, 0.6];[x(X)].
bayes n7;[0.3, 0.7];[].
```

*PFL program with Aggregation Parfactors.*

```
bayes f,f1;[1.0, 0.0, -1.0, 1.0];[].

bayes f1,e(Y);[1.0, 0.0, 1.0, 1.0];[y(Y)].

bayes e1(Y),d(Y),n1(Y);[1.0, 1.0, 1.0, 0.0,
                        0.0, 0.0, 0.0, 1.0];[y(Y)].



bayes e2(Y),d(Y),n2(Y);[1.0, 0.0, 1.0, 1.0,
                        0.0, 1.0, 0.0, 0.0];[y(Y)].
```

```
bayes e(Y),e1(Y),e2(Y);[1.0, 0.0, 0.0, 0.0,
                        0.0, 1.0, 1.0, 1.0];[y(Y)].

bayes d1(Y),c(X,Y);[1.0, 0.0, 1.0, 1.0];[x(X),y(Y)].

bayes d(Y),d1(Y);[1.0, 0.0, -1.0, 1.0];[y(Y)].


bayes c1(X,Y),b(X),n3(X,Y);[1.0, 1.0, 1.0, 0.0,
                            0.0, 0.0, 0.0, 1.0];[x(X),y(Y)].

bayes c2(X,Y),b(X),n4(X,Y);[1.0, 0.0, 1.0, 1.0,
                            0.0, 1.0, 0.0, 0.0];[x(X),y(Y)].

bayes c(X,Y),c1(X,Y),c2(X,Y);[1.0, 0.0, 0.0, 0.0,
                              0.0, 1.0, 1.0, 1.0];[x(X),y(Y)].


bayes b1(X),a,n5(X);[1.0, 1.0, 1.0, 0.0,
                     0.0, 0.0, 0.0, 1.0];[x(X)].


bayes b2(X),a,n6(X);[1.0, 0.0, 1.0, 1.0,
                     0.0, 1.0, 0.0, 0.0];[x(X)].

bayes b(X),b1(X),b2(X);[1.0, 0.0, 0.0, 0.0,
                        0.0, 1.0, 1.0, 1.0];[x(X)].


bayes a,n7;[1.0, 0.0, 0.0, 1.0];[].


bayes n1(Y);[0.9, 0.1];[y(Y)].
bayes n2(Y);[0.8, 0.2];[y(Y)].
bayes n3(X,Y);[0.7, 0.3];[x(X),y(Y)].
bayes n4(X,Y);[0.6, 0.4];[x(X),y(Y)].
bayes n5(X);[0.5, 0.5];[x(X)].
bayes n6(X);[0.4, 0.6];[x(X)].
bayes n7;[0.3, 0.7];[].
```

*WFOMC program.*

```
predicate f
predicate e(Y)
predicate d(Y)
predicate n1(Y) 0.1 0.9
predicate n2(Y) 0.2 0.8
predicate c(X,Y)
predicate b(X)
predicate n3(X,Y) 0.3 0.7
predicate n4(X,Y) 0.4 0.6
predicate a
predicate n5(X) 0.5 0.5
predicate n6(X) 0.6 0.4
predicate n7 0.7 0.3
predicate z1 1 1
predicate s1 1 -1
predicate z2(Y) 1 1
```

```
predicate s2(Y) 1 -1

f v !z1
!f v z1
z1 v !e(Y)
z1 v s1
s1 v !e(Y)

e(Y) v !d(Y) v !n1(Y)
e(Y) v d(Y) v !n2(Y)

!e(Y) v d(Y) v!d(Y)
!e(Y) v n1(Y) v!d(Y)
!e(Y) v d(Y) v n2(Y)
!e(Y) v n1(Y) v n2(Y)

d(Y) v ! z2(Y)
!d(Y) v  z2(Y)
z2(Y) v !c(X,Y)
z2(Y) v s2(Y)
s2(Y) v !c(X,Y)

c(X,Y) v !b(X) v !n3(X,Y)
c(X,Y) v b(X) v !n4(X,Y)

!c(X,Y) v b(X) v !b(X)
!c(X,Y) v n3(X,Y) v !b(X)
!c(X,Y) v b(X) v n4(X,Y)
!c(X,Y) v n3(X,Y) v n4(X,Y)

b(X) v !a v !n5(X)
b(X) v a v !n6(X)

!b(X) v a v !a
!b(X) v n5(X) v !a
!b(X) v a v n6(X)
!b(X) v n5(X) v n6(X)

a v !n7
!a v n7
```