Lifted Probabilistic Inference by First-Order Knowledge Compilation

Guy Van den Broeck, Nima Taghipour, Wannes Meert, Jesse Davis and **Luc De Raedt**
Department of Computer Science
Katholieke Universiteit Leuven
Celestijnenlaan 200A, B-3001 Heverlee, Belgium
{guy.vandenbroeck, nima.taghipour, wannes.meert, jesse.davis, luc.deraedt}@cs.kuleuven.be

Abstract

Probabilistic logical languages provide powerful formalisms for knowledge representation and learning. Yet performing inference in these languages is extremely costly, especially if it is done at the propositional level. Lifted inference algorithms, which avoid repeated computation by treating indistinguishable groups of objects as one, help mitigate this cost. Seeking inspiration from logical inference, where lifted inference (e.g., resolution) is commonly performed, we develop a model theoretic approach to probabilistic lifted inference. Our algorithm compiles a first-order probabilistic theory into a first-order deterministic decomposable negation normal form (d-DNNF) circuit. Compilation offers the advantage that inference is polynomial in the size of the circuit. Furthermore, by borrowing techniques from the knowledge compilation literature our algorithm effectively exploits the logical structure (e.g., context-specific independencies) within the first-order model, which allows more computation to be done at the lifted level. An empirical comparison demonstrates the utility of the proposed approach.

1 Introduction

Probabilistic logical languages, which combine the advantages of graphical models with those of first-order logic [Getoor and Taskar, 2007; De Raedt *et al.*, 2008], have received much attention in recent years. Since these languages offer more expressivity than their logical and probabilistic counterparts, performing efficient inference is challenging. In order to reduce the cost of inference in these languages, Poole [2003] proposed performing lifted inference, which avoids repeated computation by treating indistinguishable groups of objects as one.

Various solutions have been proposed (e.g., [de Salvo Braz *et al.*, 2005; Milch *et al.*, 2008; Jha *et al.*, 2010]). While significant progress has been made, we lack a complete understanding of how to perform lifted inference, that is, how to unify the principles of inference in first-order logic with those of inference in graphical models. The goal of lifted inference is analogous to that of the resolution principle in first-order logic as both try to perform inference at the first-order level and to ground out only when necessary.

In the literature on inference in graphical models, one can distinguish two types of approaches: *probabilistic* inference methods such as variable elimination and belief propagation and *logical* inference methods based on weighted model counting [Chavira and Darwiche, 2008]. Most existing lifted inference techniques have lifted probabilistic inference methods toward the use of first-order logic. What has been largely unexplored is applying techniques from inference in first-order logic to lifted probabilistic inference.

The key contribution of this paper is that we introduce a first-order knowledge compilation approach to lifted probabilistic inference. We do so by upgrading propositional d-DNNF (deterministic decomposable negation normal form) circuits to the first-order setting. Given a weighted clausal theory (i.e., weights are assigned to predicates), our approach compiles it into a first-order d-DNNF, which can perform weighted model counting in time polynomial in the size of the first-order d-DNNF. It is easy to show that several existing probabilistic logic representations such as parfactors [Poole, 2003] and Markov Logic [Singla and Domingos, 2008] can be transformed into an equivalent weighted first-order theory.

In this paper we present a *model theoretic* approach to lifted inference that builds on well known concepts from logical inference, such as model counting, unit propagation, and Shannon decomposition. We show essentially that existing concepts from lifted probabilistic inference have their model-theoretic counterpart. Furthermore, our WFOMC (weighted first-order model counting) approach exploits context-specific independencies [Boutilier *et al.*, 1996], which arise when the probabilistic dependencies have an internal logical structure. We also show that WFOMC can lift more inference problems than alternative approaches.

2 Background and Notation

We first introduce standard notions of (function free) first-order logic. An *atom* $p(t_1, \ldots, t_n)$ consists of a predicate p/n of arity n followed by n terms t_i. *Terms* are either *constants* or *variables*. A *literal* l is an atom a or its negation $\neg a$. A *clause* is a disjunction $l_1 \vee ... \vee l_k$ of literals. If $k = 1$, we have a *unit* clause. We often write clauses as their set of literals. Clauses are assumed to be universally quantified. A *substitution* $\theta = \{v_1 = t_1, ..., v_l = t_l\}$ is an assignment of

terms t_i to variables v_i. A *theory* is a finite set of clauses denoting the conjunction of its clauses. An *expression* is an atom, literal, clause or theory. The set of all logical variables appearing in an expression e is denoted by $\mathrm{Vars}(e)$. The functions $\mathrm{atom}(e)$ and $\mathrm{lit}(e)$ map an expression to the atoms and literals it contains. An expression is *ground* if it does not contain any variables. A Herbrand *interpretation* is a set of ground atoms. All atoms in the interpretation are assumed to be true, while all other atoms belonging to the Herbrand base (the set of all possible ground atoms) are assumed to be false. An interpretation I satisfies a theory T, written as $I \models T$, if it satisfies all the clauses $c \in T$. An interpretation that satisfies the theory is a *model* for the theory and we denote the set of models of a theory T by $\mathcal{M}(T) = \{I \mid I \models T\}$.

We now extend these notions by adding to each expression e a *constraint set* (CS) which is a set of constraints defined on $\mathrm{Vars}(e)$. The CS essentially represents a constraint satisfaction problem and denotes a conjunction of constraints of the form: 1) $v \in D$, where D is the domain of the variable v and $v \in \mathrm{Vars}(e)$; 2) $v_i \neq v_j$ with $v_i, v_j \in \mathrm{Vars}(e)$; 3) or, $v \neq k$, with $v \in \mathrm{Vars}(e)$ and k a constant. Negations of these constraints are also allowed. A substitution $\theta = \{v_1 = t_1, \ldots, v_l = t_l\}$ is a solution to a CS cs if and only if all constraints in $cs\theta$ are ground and satisfied. When a solution to a CS exists, we say the CS is satisfiable. The set of assignments to variable v for which the CS is satisfiable is denoted by $\mathrm{Sol}(cs, v)$.

Expressions e together with their associated constraint set cs are called *c-expressions* and are written as (e, cs). Thus, we can talk about *c-literals*, *c-clauses* and *c-theories*. The functions $\mathrm{atom_c}(e)$, $\mathrm{lit_c}(e)$ and $\mathrm{cs}(e)$ map an expression to its c-atoms, c-literals and CS. In the remainder of the paper we work with c-expressions and often drop the c for ease of exposition. Furthermore, constraints can be simplified or reduced using standard constraint solving techniques and we implicitly allow that. Space restrictions do not allow for a more detailed analysis. One particular type of reduction is to apply equalities of the form $v = t$, i.e., a c-expression $(e, \{cs, v = t\})$ is reduced to $(e\{v = t\}, cs\{v = t\})$. A substitution $\theta = \{v_1 = t_1, \ldots, v_l = t_l\}$ can be regarded as a conjunction of equality constraints, e.g., $v_1 = t_1 \wedge \ldots \wedge v_l = t_l$. Therefore, applying a substitution θ to a c-expression (e, cs) yields the c-expression $(e, cs \wedge \theta)$ which is equivalent to $(e\theta, cs\theta)$.

Two c-literals $L_1 = (l_1, cs_1)$ and $L_2 = (l_2, cs_2)$ *c-unify* with substitution θ if and only if $l_1\theta = l_2\theta$ and the CS $(cs_1 \wedge cs_2)\theta$ is satisfiable. The result is denoted as $(l_1\theta, (cs_1 \wedge cs_2)\theta)$, or $\theta = (l_2\theta, (cs_1 \wedge cs_2)\theta)$. We write that $L_1\theta \simeq L_2\theta$. The *most general unifier* is denoted by $\theta = \mathrm{mgu}(L_1, L_2)$. Two c-expressions e_1 and e_2 are *independent*, written as $e_1 \perp\!\!\!\perp e_2$, if no c-atom $a_1 \in \mathrm{atom_c}(e_1)$ c-unifies with a c-atom $a_2 \in \mathrm{atom_c}(e_2)$. The set of groundings $\mathrm{gr}(e)$ of a c-expression e is the set of ground c-expressions e_g s.t. $\exists\theta$ for which $e_g = e\theta$ and $\mathrm{cs}(e)\theta$ is satisfied. Intuitively, a clause c represents all its ground instances $c\theta$, whereas a constrained clause (c, cs) represents all ground instances $c\theta$ such that $cs\theta$ is true according to the constraint theory.

Example 1. Take the constrained literal

$$l = (\mathrm{flies}(X), X \neq \mathit{kiwi} \wedge X \in \mathit{Bird}).$$

The mgu of l and $\mathrm{flies}(\mathit{pigeon})$ is $\theta = \{X = \mathit{pigeon}\}$, because it is the most general unifier of the unconstrained literals and $\mathit{csp}(l)\theta = (\mathit{pigeon} \neq \mathit{kiwi} \wedge \mathit{pigeon} \in \mathit{Bird})$ is satisfied. Constrained literal l is independent from $\mathrm{flies}(\mathit{kiwi})$ because the mgu of the unconstrained literals gives rise to the unsatisfied constraint $\mathit{kiwi} \neq \mathit{kiwi}$.

3 Algorithm

Our approach to lifted inference converts the first-order probabilistic inference task to a weighted first-order model counting problem. We do so by compiling a clausal theory to a first-order d-DNNF circuit, thereby lifting the knowledge compilation approach. In the next subsections we discuss the key elements of our approach: (i) how to cast probabilistic inference into a weighted model counting problem, (ii) which formula does a circuit represent, (iii) how to compile the theory to an equivalent circuit, and (iv) how to compute the weighted model count using the circuit.

3.1 Weighted First-Order Model Counting

Weighted First-Order Model Counting (WFOMC) augments a theory in first-order logic with a positive weight function w and a negative weight function $\overline{\mathrm{w}}$, which assign a weight to each predicate. The WFOMC problem for a theory T of c-clauses consists of computing

$$\mathrm{wmc}(T, \mathrm{w}, \overline{\mathrm{w}}) = \sum_{I \models T} \prod_{a \in I} \mathrm{w}(\mathrm{Pred}(a)) \prod_{a \in HB(T) \setminus I} \overline{\mathrm{w}}(\mathrm{Pred}(a))$$

where $HB(T)$ is the Herbrand base and Pred maps atoms to their predicate.

Representing existing probabilistic logical models such as Markov logic networks (MLN) or parfactors as WFOMC problems requires transforming them into a set of c-clauses.

Example 2. An MLN is a set of pairs (F_i, r_i) where F_i is a formula in first-order logic and r_i a real-valued number. Since WFOMC works on literals and not clauses, we reify each formula F_i in the MLN, creating a new atom $\mathrm{f}_i(\mathrm{Vars}(F_i)) \equiv F_i$. The weight functions of the reified atom are $\mathrm{w}(\mathrm{f}_i) = e^{r_i}$ and $\overline{\mathrm{w}}(\mathrm{f}_i) = 1$.

Take the MLN formula with weight 2:

$$2\ \mathrm{friends}(X, Y) \wedge \mathrm{smokes}(X) \Rightarrow \mathrm{smokes}(Y).$$

We introduce a new predicate f that represents the truth value of this formula and which is assigned $\mathrm{w}(\mathrm{f}) = e^2$ and $\overline{\mathrm{w}}(\mathrm{f}) = 1$. The friends and smokes predicates are assigned $\mathrm{w}(.) = \overline{\mathrm{w}}(.) = 1$. In the logical theory we put

$$\mathrm{f}(X, Y) \equiv [\mathrm{friends}(X, Y) \wedge \mathrm{smokes}(X) \Rightarrow \mathrm{smokes}(Y)]$$

which in CNF form is

$$\begin{array}{l} \mathrm{smokes}(Y) \vee \neg\,\mathrm{smokes}(X) \vee \neg\,\mathrm{friends}(X, Y) \vee \neg\,\mathrm{f}(X, Y) \\ \mathrm{friends}(X, Y) \vee \mathrm{f}(X, Y) \\ \mathrm{smokes}(X) \vee \mathrm{f}(X, Y) \\ \neg\,\mathrm{smokes}(Y) \vee \mathrm{f}(X, Y). \end{array}$$

The WFOMC of this theory is equal to the partition function of the MLN. Parfactors can be transformed into a set of c-clauses in a similar way.

3.2 First-Order d-DNNFs

Deterministic decomposable negation normal form (d-DNNF) circuits [Darwiche and Marquis, 2002] are directed acyclic graphs whose leaves represent literals and whose inner nodes represent formulae in propositional logic. The inner node operators in propositional d-DNNFs are:

- Decomposable conjunction $l \owedge r$, representing the formula $l \wedge r$ with the constraint that the operand formulas l and r are independent ($l \perp\!\!\!\perp r$).
- Deterministic disjunction $l \ovee r$, representing the formula $l \vee r$ with the constraint that the operands cannot be true at the same time ($l \wedge r$ is unsatisfiable).

We generalize propositional d-DNNFs to first-order d-DNNF (FO d-DNNF) circuits, where leaves represent first-order literals and inner nodes represent first-order formulae. FO d-DNNFs are more compact than their propositional counterparts. In addition to the propositional operators in inner nodes, we define the following novel operators:

- Inclusion-exclusion $\mathrm{IE}(x, y, z)$, representing the formula $x \vee y$ with the extra operand $z \equiv x \wedge y$, which is required to perform (weighted) model counting in the circuit.
- Decomposable set-conjunction $\owedge c$ over isomorphic operands, represented by a single child circuit c.
- Deterministic set-disjunction $\ovee c$ over isomorphic operands, represented by a single child circuit c.

These last two operators differ from their propositional counterparts in that they have only one child node c, which represents a potentially large set of isomorphic operands.

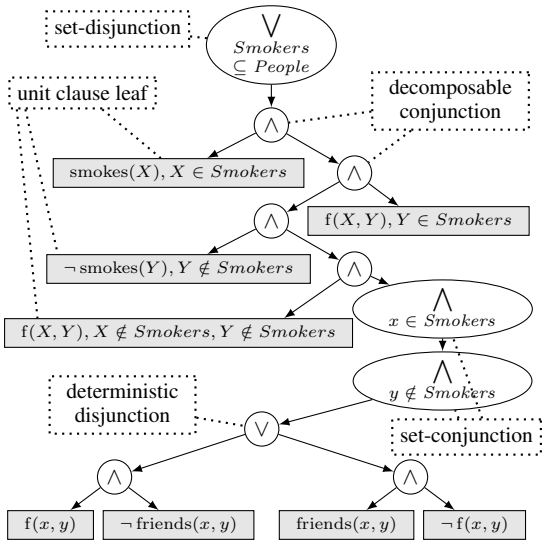


Figure 1: FO d-DNNF circuit of Example 2.

Example 3. Figure 1 shows the FO d-DNNF circuit for the CNF of Example 2. Variables are in the domain $People$.

Example 4. Figure 2b represents a set-conjunction of $|People|$ operands. Since all operands are isomorphic (identical up to the value of Skolem constant x), they are represented by a single child, parametrized in x. We will later see that these operands have identical weighted model counts.

Similarly, Figure 2c shows a set-disjunction of theories that are identical up to the value of $FunPeople$. Again, the circuit only contains a single child of the set-disjunction, which is parametrized in $FunPeople$.

3.3 Auxiliary Operations

Next, we introduce the auxiliary operations for splitting and shattering, which are needed in the compilation process. Whenever the pre-condition holds before applying the operation, the post-condition holds after applying it.

Splitting

Some compilation rules require that all c-clauses c in the theory be split w.r.t. a certain c-atom a.

Operator $\textsc{Split}(c, a) =$
if $\forall a_c \in \mathrm{atom_c}(c) : a_c \perp\!\!\!\perp a \vee \mathrm{gr}(a_c) \setminus \mathrm{gr}(a) = \emptyset$ **then** $\{c\}$
else $\{\textsc{Split}(c_{mgu}, a)\} \cup \bigcup_{c_r \in C_r} \textsc{Split}(c_r, a)$
for some $a_c \in \mathrm{atom_c}(c)$ such that $\mathrm{gr}(a_c) \setminus \mathrm{gr}(a) \neq \emptyset$,
$\theta = \mathrm{mgu}(a, a_c)$,
$c_{mgu} = (\mathrm{lit}(c), \mathrm{cs}(c) \wedge \theta \wedge \mathrm{cs}(a))$,
and $C_r = \{(\mathrm{lit}(c), \mathrm{cs}(c) \wedge \neg e) | e \in (\theta \wedge \mathrm{cs}(a))\} \setminus \{c\}$

Postconditions

1. $\forall a_s \in \mathrm{atom_c}(\textsc{Split}(c, a)) : a_s \perp\!\!\!\perp a \vee \mathrm{gr}(a_s) \subseteq \mathrm{gr}(a)$
2. $\mathrm{gr}(c) = \bigcup_{c_s \in \textsc{Split}(c,a)} \mathrm{gr}(c_s)$

The purpose of splitting is to divide a clause into an equivalent set of clauses (Postcondition 2) such that for each atom a_s in each clause, either a_s is independent from a or is subsumed by it, because it covers a subset of the ground atoms of a (Postcondition 1).

Example 5. Splitting

$c = (\mathrm{flies}(X) \vee \neg\,\mathrm{haswings}(X), X \neq \mathit{kiwi} \wedge X \in \mathit{Animal})$
$a = (\mathrm{flies}(X), X \neq \mathit{penguin} \wedge X \in \mathit{Bird})$,

where domain $\mathit{Bird} \subset \mathit{Animal}$, results in $c_{mgu} = (\mathrm{flies}(X) \vee \neg\,\mathrm{haswings}(X), X \neq \mathit{kiwi} \wedge X \neq \mathit{penguin} \wedge X \in \mathit{Bird})$, $c_r^1 = (\mathrm{flies}(\mathit{penguin}) \vee \neg\,\mathrm{haswings}(\mathit{penguin}))$ and $c_r^2 = (\mathrm{flies}(X) \vee \neg\,\mathrm{haswings}(X), X \neq \mathit{kiwi} \wedge X \in \mathit{Animal} \setminus \mathit{Bird})$.

After splitting, clauses c_r^1 and c_r^2 are independent from a, while the c-atom $(\mathrm{flies}(X), X \neq \mathit{kiwi} \wedge X \neq \mathit{penguin} \wedge X \in \mathit{Bird})$ in c_{mgu} is implied by a.

Finally, splitting an entire theory C with respect to c-atom a_c is defined as $\textsc{Split}(C, a_c) = \bigcup_{c \in C} \textsc{Split}(c, a_c)$. Splitting was introduced by Poole [2003] for parfactors. We apply it to clauses and extend it with set membership constraints.

Shattering

Some compilation rules require the theory C to be shattered.

Operator $\textsc{Shatter}(C) =$
if $\exists a \in \mathrm{atom_c}(C)$ **such that** $\textsc{Split}(C, a) \neq C$
then $\textsc{Shatter}(\textsc{Split}(C, a))$ **else** C

Postcondition $\forall a_1, a_2 \in \mathrm{atom_c}(\textsc{Shatter}(C)) :$
$a_1 \perp\!\!\!\perp a_2 \vee \mathrm{gr}(a_1) = \mathrm{gr}(a_2)$

Shattering performs splitting until convergence. The postcondition states that after shattering, all c-atom groundings are either disjoint or identical. The specific assignments to the logical variables cannot be distinguished any further. A variant of shattering on parfactors was proposed for FOVE [de Salvo Braz *et al.*, 2005].

3.4 Compilation Rules

The purpose of the compilation rules is to transform a CNF into a set of simplified CNFs that are combined using a FO d-DNNF operator from Section 3.2. This requires that (1) the compiled circuit is equivalent to the original theory and (2) the (decomposable, deterministic) properties of the FO d-DNNF operators hold. We continue compiling the simplified CNFs until they become unit clauses, the leafs of the FO d-DNNF circuit.

Compiling a set of c-clauses C to FO d-DNNF is performed by the $\text{COMPILE}(C)$ function, which tries to recursively apply the compilation rules in the order we present them here. A compilation rule applies when its preconditions are met.

Caching

Before compiling a theory, we attempt to retrieve its circuit from a cache of previously compiled theories. Reusing previously computed FO d-DNNFs turns the circuit from a tree into a DAG. Caching is an essential feature of propositional knowledge compilation algorithms. Currently, we employ a naive caching scheme that only recognizes the reuse of ground theories.

Unit Propagation

We now consider the case where the CNF contains unit clauses. First, we describe an auxiliary operation to perform unit propagation of a c-literal l on a single c-clause c, where a is the atom of l.

Precondition $\{c\} = \text{SPLIT}(c, a)$ (c has been split w.r.t. a)
Operator $\text{UPCLAUSE}(c, l) =$
if $\exists\, l_c \in \text{lit}_\text{c}(c), \exists\, \theta : l\theta \simeq l_c\theta$ **then** $\emptyset$
else $\{(L, \text{cs}(c))\}$ with $L = \{l_c | l_c \in \text{lit}_\text{c}(c) : l \perp\!\!\!\perp l_c\}$

Unit propagation distinguishes between two cases: if the clause contains a literal that unifies with the propagating literal l, the clause is removed. Otherwise, the clause atoms that unify with the propagating atom are removed from the clause.

This generalizes to a compilation rule on a set of clauses C containing a unit clause c_u.

Precondition $\exists\, c_u \in C : \text{lit}_\text{c}(c_u) = \{l\}, \text{atom}_\text{c}(c_u) = \{a\}$
Operator $\text{UNITPROP}(C) =$
$\text{COMPILE}(C_{up}) \circledwedge \text{COMPILE}(\{c_u\})$
where $C_{up} = \bigcup_{c \in C} \bigcup_{s \in \text{SPLIT}(c,a)} \text{UPCLAUSE}(s, l)$
Postcondition $C_{up} \perp\!\!\!\perp c_u$ and $C \equiv C_{up} \wedge c_u$

The Postcondition states that the returned conjunction is decomposable and equivalent to the original theory.

Example 6. Figure 2a shows unit propagation of the unit clause $\text{friends}(X, X)$. The first two clauses require splitting w.r.t. the unit clause atom, which creates two copies of each clause: one where $X = Y$ and one where $X \neq Y$. Next, the first copy of the first clause is removed, because it is subsumed by the unit clause. In the first copy of the second clause, the atom that unifies with the unit clause atom is removed, because it must be false in any model. This yields the clause $\text{likes}(X, X)$.

Finally, the unit clause becomes independent of the other clauses, and can be split off in a decomposable conjunction. The unit clause is a leaf of the circuit, while the remainder of the theory requires further compilation.

Independence

When the theory contains two sets of independent clauses, we can split them using a decomposable conjunction.

Precondition $C_1 \perp\!\!\!\perp C_2$
Operator
$\text{INDEP}(C_1 \cup C_2) = \text{COMPILE}(C_1) \circledwedge \text{COMPILE}(C_2)$

Inclusion-Exclusion

Intuitively, when the theory contains a clause whose literals can be divided into two sets that do not share any logical variables or constraints, inclusion-exclusion is possible. The children of an inclusion-exclusion node are simpler than their parent, because they have shorter clauses.

Preconditions C is a set of c-clauses such that
1. $C = \{(L_1 \cup L_2, cs_L)\} \cup C_R$
2. $\text{Vars}(L_1) \cap \text{Vars}(L_2) = \emptyset$
3. $\neg\exists v_1 \in \text{Vars}(L_1), v_2 \in \text{Vars}(L_2) : (v_1 \neq v_2) \in cs_L$

Operator $\text{INCEXC}(C) =$
$\text{IE}(\text{COMPILE}(C_1), \text{COMPILE}(C_2), \text{COMPILE}(C_3))$
with $C_1 = \{(L_1, cs_L)\} \cup C_R$
$C_2 = \{(L_2, cs_L)\} \cup C_R$
$C_3 = \{(L_1, cs_L), (L_2, cs_L)\} \cup C_R$.
Postconditions
1. $\mathcal{M}(C) = \mathcal{M}(C_1) \cup \mathcal{M}(C_2)$
2. $\mathcal{M}(C_3) = \mathcal{M}(C_1) \cap \mathcal{M}(C_2)$.

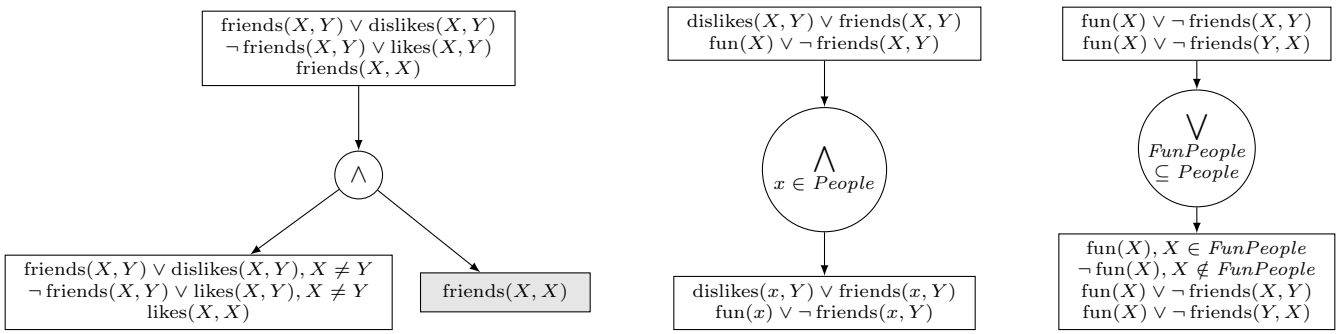
Example 7. C-clause $(\text{helps}(X) \vee \neg\,\text{succeeds}(Y), X \in \mathit{People}, Y \in \mathit{Task})$ has two subclauses: $c_1 = (\text{helps}(X), X \in \mathit{People})$ and $c_2 = (\neg\,\text{succeeds}(X), X \in \mathit{Task})$ which are not linked by a logical variable. The INCEXC operator results in an inclusion-exclusion node with child circuits for c_1, c_2 and $c_1 \wedge c_2$.

A special case of inclusion-exclusion, *Shannon decomposition*, occurs when $L_1 = \{a\}$ and $L_2 = \{\neg a\}$ and a is a ground atom. Then C_3 becomes a contradiction and $C_1 \vee C_2$ is a deterministic disjunction. This decomposition is always performed when the theory contains a ground atom, because adding the clause $a \vee \neg a$ to a theory containing a does not change the set of models. Shannon decomposition is a common operation in logical inference. We generalize it to inclusion-exclusion.

Example 8. Having a c-clause $c = (\text{fun}(\mathit{bob}) \vee \neg\,\text{friends}(\mathit{bob}, X), X \in \mathit{People})$, applying Shannon decomposition results in a decomposable disjunction of the circuits for $\{\{\text{fun}(\mathit{bob})\}, c\}$ and $\{\{\neg\,\text{fun}(\mathit{bob})\}, c\}$, which after an application of UNITPROP results in $\text{fun}(\mathit{bob}) \circledvee (\neg\,\text{fun}(\mathit{bob}) \circledwedge \neg\,\text{friends}(\mathit{bob}, X), X \in \mathit{People})$.

Shattered Compilation

For the next two operations, the theory needs to be shattered. Furthermore, shattering might introduce new opportunities for the independence, inclusion-exclusion or Shannon decomposition rules. Because we only shatter in this stage of the compilation process, the previous operators use what is called *splitting as needed* [Kisyński and Poole, 2009].



(a) Unit Propagation of $\text{friends}(X, X)$ (b) Independent Partial Grounding (c) Atom Counting of $\text{fun}(X)$

Figure 2: Intermediate compilation steps. Circles are FO d-DNNF inner nodes. White rectangles show the theories before and after applying the rule. Shaded rectangles are leafs of the circuit, representing unit clauses. All variable domains are $People$.

Preconditions C is a set of clauses

1. $\exists\, a \in \text{atom}_\text{c}(C) : \textsc{Split}(C, a) \neq C$

Operator $\textsc{ShatterComp}(C) = \textsc{Compile}(\textsc{Shatter}(C))$

Independent Partial Groundings

Preconditions C is a set of c-clauses such that

1. $\neg\, \exists\, C_I \subset C : C_I \perp\!\!\!\perp C \setminus C_I$
2. A function root such that $\forall\, c \in C : \text{root}(c) \in \text{Vars}(c)$
3. $\forall\, l_1, l_2 \in \text{lit}_\text{c}(C) : l_1\theta_{k_1} \perp\!\!\!\perp l_2\theta_{k_2}$
   where $\theta_{k_i} = \{(\text{root}(c) = k_i) | c \in C\}$
   and k_1, k_2 arbitrary unique constants

Operator $\textsc{Ipg}(C) = \bigwedge_{k \in K} \textsc{Compile}(C\theta_k)$
with k a parameter for a constant,
and K such that $\forall\, c \in C : K = \text{Sol}(\text{root}(c), \text{cs}(c))$

Postconditions

1. $C\theta_{k_1} \perp\!\!\!\perp C\theta_{k_2}$ with k_1, k_2 arbitrary unique constants
2. $C \equiv \bigwedge_{k \in K} C\theta_k$
3. $\forall\, k_1, k_2 \in K : \textsc{Compile}(C\theta_{k_1}) \cong \textsc{Compile}(C\theta_{k_2})$,
   meaning that all operand circuits are isomorphic.

Precondition 3 guarantees that every literal in the theory contains a variable in $\{\text{root}(c) | c \in C\}$ and that the variable appears in the same positions of the argument lists of two unifying atoms. Consequently, any pair of atoms that previously unified, becomes independent after partially grounding to a different constant. Postcondition 2 states that the set-conjunction of all operands is equivalent to the original theory. Postcondition 1 makes the set-conjunction decomposable and follows straightforwardly from Precondition 3.

The operands of a decomposable set-conjunction were defined to be isomorphic. If after shattering the theory contains no independent sets of clauses (Precondition 1), the constraints and domains of each $\text{root}(c)$ are identical and each clause is grounded w.r.t. the same set of constants K. Applying independent partial grounding in a naive way would create $|K|$ subcircuits – a potentially very large number. However, because the subcircuits are isomorphic (identical up to renaming of the grounding constant, Postcondition 3), the d-DNNF for C is more succinctly represented by compiling only one child theory $C\theta_k$, where k is a parameter representing the grounding constant. This makes both the compilation complexity and FO d-DNNF size independent from $|K|$. Independent partial grounding is based on the same underlying ideas as partial inversion in FOVE and the power rule in CPs [Jha *et al.*, 2010].

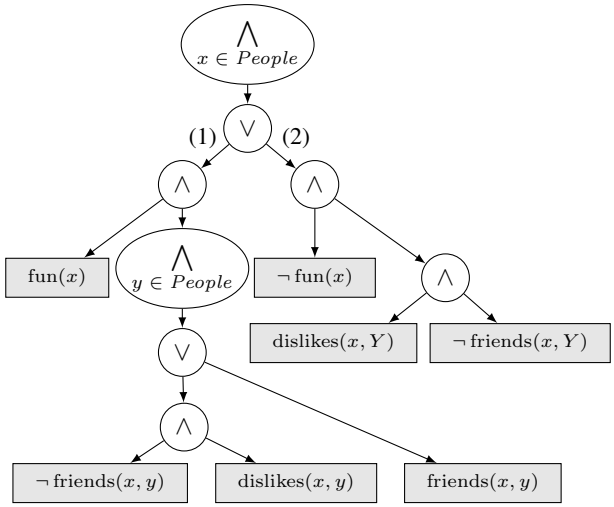


Figure 3: Circuit after continuing compilation of Figure 2b

Example 9. Figure 2b shows a theory T where independent partial grounding can be applied by choosing the root function that maps each clause to its X variable. When these variables are grounded to different constants ($T\theta_{alice}$ and $T\theta_{bob}$), the different groundings are independent, which means they form a decomposable conjunction. This naive approach would generate $|People|$ subcircuits. We can do better by observing that the partially grounded theories are identical up to renaming of the constants, and so are the circuits. Therefore, it suffices to compile a single theory $T\theta_x$ where x represents some constant in the domain $People$.

Independent partial grounding does not simplify the theory. By grounding logical variables, however, it creates new opportunities for other rules. For example, the bottom the-

ory of Figure 2b contains a ground atom $\mathrm{fun}(x)$, which allows for Shannon decomposition. This is shown in Figure 3. In the branch where $\mathrm{fun}(x)$ is false (labeled with (2) in Figure 3), subsequent steps of unit propagation simplify the theory in a decomposable conjunction of unit clause leafs. The other branch (1) requires one more step of independent partial grounding and Shannon decomposition.

Atom Counting

Preconditions C is a set of c-clauses such that

1. $\exists\, a \in \mathrm{atom_c}(C) : \mathrm{Vars}(a) = \{v\}$

Operator $\textsc{AtomCounting}(C) =$
$\bigvee_{D \subseteq S} \textsc{Compile}(C \cup \{c_D, \bar{c}_D\})$
with $S = \mathrm{Sol}(v, \mathrm{cs}(a))$, D the domain parameter,
$c_D = (\{a\}, \mathrm{cs}(a) \wedge (v \in D))$,
and $\bar{c}_D = (\{\neg a\}, \mathrm{cs}(a) \wedge (v \notin D))$.

Postconditions For $F, G \subseteq S$:

1. $\mathcal{M}(C \cup \{c_F, \bar{c}_F\}) \cap \mathcal{M}(C \cup \{c_G, \bar{c}_G\}) = \emptyset \vee F = G$
2. $\textsc{Compile}(C \cup \{c_F, \bar{c}_F\}) \cong \textsc{Compile}(C \cup \{c_G, \bar{c}_G\})$, meaning that all operand circuits are isomorphic.

When the theory contains an atom with a single logical variable v, we can partition the set of models depending on the values for v for which the atom is true. The disjunctions between these partitions are deterministic (Postcondition 1), because their models disagree in at least one atom.

Because the theory has been shattered, the individuals $k \in D$ are indistinguishable. As a result, the operands of the deterministic disjunction are isomorphic (Postcondition 2) and their circuits only differ in the domain D. Again, we only compile a FO d-DNNF that is parametrized in D but whose size is independent of $|D|$ and $|S|$. Atom counting by itself does not simplify the theory, but by introducing two new unit clauses, unit propagation will be able to eliminate a from the theory entirely in subsequent compilation steps. Atom counting is inspired by counting elimination in FOVE and the generalized binomial rule in CPs.

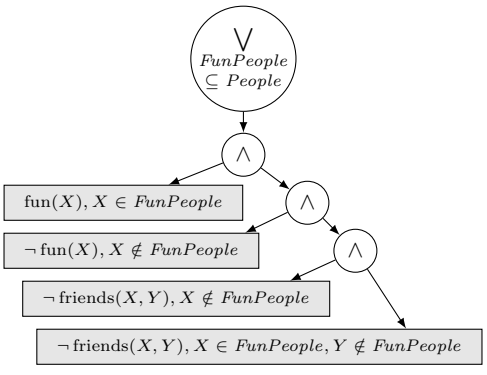


Figure 4: Circuit after continuing compilation of Figure 2c

Example 10. Figure 2c shows a theory T that only allows for atom counting. All logical variables are implicitly elements of domain $People$. The $\mathrm{fun}(X)$ atom has $2^{|People|}$ partial interpretations (e.g., $\{\mathrm{fun}(1), \mathrm{fun}(2)\}$, $\{\mathrm{fun}(1), \neg\,\mathrm{fun}(2)\}$, …, $\{\neg\,\mathrm{fun}(1), \neg\,\mathrm{fun}(2)\}$ when the number of people is 2). We can create a theory equivalent to T by conjoining each partial interpretation with T, and taking the disjunction over all partial interpretations. Because each partial interpretation differs in at least one truth assignment, this disjunction is deterministic.

Compiling the theory as such defies the point of lifted inference, because it creates $2^{|People|}$ subcircuits. However, we can observe that all subcircuits are isomorphic. They are identical up to the set of constants c for which $\mathrm{fun}(c)$ is true. We only need to compile one single subcircuit, parametrized in the subset $FunPeople$ of $People$. We will later see that evaluating the weighted model count using this circuit is only linear in $|People|$, which makes this operator lifted.

Atom counting only adds unit clauses to the theory. Figure 4 shows how three subsequent steps of unit propagation split off a unit clause leaf until the remaining theory becomes a unit clause leaf of the circuit itself.

Grounding

Operator $\textsc{Gr}(C) = \textsc{Compile}(\cup_{k \in \mathrm{Sol}(v, \mathrm{cs}(v))} C\{v = k\})$
for some $v \in \mathrm{Vars}(C)$

If no other compilation rule applies, we ground out a logical variable in the theory, after which compilation is retried. A good heuristic is to select the variable with the smallest number of solutions $|\,\mathrm{Sol}(v, \mathrm{cs}(v))|$.

3.5 First-Order Smoothing

Compilation steps such as unit propagation or inclusion-exclusion may remove literals from branches of the d-DNNF. The groundings of these literals may go unaccounted for when doing model counting in the d-DNNF. This problem is solved by *smoothing* the circuit [Darwiche and Marquis, 2002], which we lift to the first-order level as well.

Smoothing first propagates a set of c-atoms upwards in the circuit, representing the groundings that are accounted for in the circuit. Binary conjunctions, disjunctions and inclusion-exclusion nodes propagate the union of these sets. In set-conjunctions with a parametrized constant k_r, special care is taken to generalize the counted atoms of the child to the full set of operands by applying the inverse substitution $\theta_{k_r}^{-1}$, which replaces the placeholder constant k_r by the variables to which the root function maps. Similarly, for set-disjunctions parametrized in domain D, the set of counted atoms of the representative child is generalized to the full set of operands by removing the domain constraints $v \in D$ and $v \notin D$ from the atoms.

Second, smoothing adds nodes to the circuit where operands of (set-)$\bigvee$ or IE count different sets of atoms. When an atom a with constraints cs_a is missing from node n, the node is replaced by $\textsc{Compile}(\{a, \neg a\}, cs_a) \bigwedge n$

Example 11. The circuit in Figure 3 is not smooth. The bottom disjunction's operands cover a different set of atoms. The right branch covers only $\mathrm{friends}(x, y)$ while the left branch covers $\mathrm{friends}(x, y)$ and $\mathrm{dislikes}(x, y)$. In this case, first-order smoothing is identical to propositional smoothing. It substitutes the right branch by a decomposable conjunction of the right branch and the circuit $\mathrm{dislikes}(x, y) \bigvee \neg\,\mathrm{dislikes}(x, y)$.

Example 12. The circuit in Figure 4 is not smooth. The theory in the root node (Figure 2c) covers atoms $\mathrm{fun}(X)$ and $\mathrm{friends}(X, Y)$. The circuit below the root node covers $\mathrm{fun}(X)$ entirely, but is not accounting for the atoms $\mathrm{friends}(X, Y), X \in \mathit{FunPeople}, Y \in \mathit{FunPeople}$. First-order smoothing compensates for this by inserting a decomposable conjunction below the set-disjunction. The operands of the new conjunction are (1) the original child of the set-disjunction and (2) the compiled circuit for the theory $\mathrm{friends}(X, Y) \vee \neg\,\mathrm{friends}(X, Y), X \in \mathit{FunPeople}, Y \in \mathit{FunPeople}$. As a result, all ground atoms of the original theory are being counted.

3.6 Propagating the Weighted Model Count

As for propositional d-DNNFs, smooth FO d-DNNFs permit weighted model counting in time polynomial in the size of the circuit, by interpreting the d-DNNF as an arithmetic circuit. In a leaf with c-atom a of predicate p:

$$\mathrm{wmc}(a) = \mathrm{w}(p)^{|\,\mathrm{gr}(a)|} \quad \text{and} \quad \mathrm{wmc}(\neg a) = \overline{\mathrm{w}}(p)^{|\,\mathrm{gr}(a)|}.$$

As in propositional knowledge compilation and from Post-conditions 1 and 2 of IncExc:

$$\begin{aligned} \mathrm{wmc}(l \mathbin{\textcircled{\wedge}} r) &= \mathrm{wmc}(l) \times \mathrm{wmc}(r), \\ \mathrm{wmc}(l \mathbin{\textcircled{\vee}} r) &= \mathrm{wmc}(l) + \mathrm{wmc}(r), \\ \mathrm{wmc}(\mathrm{IE}(c_1, c_2, c_3)) &= \mathrm{wmc}(c_1) + \mathrm{wmc}(c_2) - \mathrm{wmc}(c_3). \end{aligned}$$

For a set-conjunction coming from independent partial grounding, the operands are not only isomorphic, but they have the same WFOMCs, due to shattering. Its weighted model count is $\mathrm{wmc}(\textcircled{\bigwedge}_{k \in K} \mathit{child}_k) = \mathrm{wmc}(\mathit{child}_k)^{|K|}$, which can be computed in time independent of $|K|$.

Set-disjunction due to atom counting is more complicated. We know that the operands have d-DNNF circuits isomorphic to child_D, but they have different weighted model counts. Iterating over all $D \subseteq S$ would be intractable and defeats the purpose of lifted inference. However, with $\mathcal{D}_n = \{D | D \subseteq S, |D| = n\}$, we observe that $\forall D_1, D_2 \in \mathcal{D}_n : \mathrm{wmc}(\mathit{child}_{D_1}) = \mathrm{wmc}(\mathit{child}_{D_2})$. We can group together subsets of S of equal size, because they have identical WFOMCs. Furthermore, because set-disjunction is deterministic and $|\mathcal{D}_n| = \binom{|S|}{n}$, we find

$$\mathrm{wmc}(\textcircled{\bigvee}_{D \subseteq S} \mathit{child}_D) = \sum_d \binom{|S|}{d} \mathrm{wmc}(\mathit{child}_D \wedge |D| = d).$$

While model counting for all other node types is independent of the domain sizes, model counting for set-disjunction is linear in $|S|$.

4 Empirical Evaluation

In this section, we evaluate our approach on common benchmarks in the lifted inference literature including *competing workshops* and *workshop attributes* [Milch *et al.*, 2008], *friends and smokers* [Singla and Domingos, 2008], *sick and death* [de Salvo Braz *et al.*, 2005] as well as a novel example. We compare the performance of WFOMC with counting first-order variable elimination (C-FOVE) [Milch *et al.*, 2008] and propositional variable elimination (VE). WFOMC is implemented in Scala[1] and we used the publicly available Java implementation of C-FOVE.[2]

Figure 5 contains representive results on three different tasks. It plots how inference time varies with the domain size on each task. It includes two results for WFMOC, one which only measures circuit evaluation time and another that includes both compilation and evaluation time. Compilation is only needed once per theory and can be amortized across all domain sizes. For C-FOVE and ground VE, the plot shows inference time as these methods have no compilation phase.

For both the *competing workshops* and *friends and smokers* tasks, ground VE quickly runs out of memory and achieves worse performance than the lifted methods. When considering only inference time, WFOMC is the same or faster than C-FOVE. When considering both compilation and inference time, WFOMC is faster for larger domain sizes, but C-FOVE is slightly faster for small domain sizes.

We extended the friends and smokers example with an extra formula: $\mathrm{friends}(X, Y) \wedge \mathrm{drinker}(X) \Rightarrow \mathrm{drinker}(Y)$. WFOMC can lift this theory whereas C-FOVE cannot lift it. We omit the curve for C-FOVE because the implementation fails.[3] Ground VE quickly runs out of memory.

Lifted inference provides a significant advantage compared to ground inference. WFOMC can identify and lift more structures than C-FOVE making it more efficient in these cases. In cases where the operators in C-FOVE are sufficient to fully lift the theory, WFOMC's inference is faster, but has a small overhead associated with the compilation step. It is important to note that knowledge compilation only needs to be performed once per theory, that is, it is independent of the domain size (for a fixed evidence set).

5 Related Work

WFOMC builds upon the technique of probabilistic inference through knowledge compilation [Darwiche and Marquis, 2002]. To date, these approaches have only explored compiling propositional d-DNNF circuits [Chavira *et al.*, 2006]. In contrast, we define first-order d-DNNF circuits and present an algorithm for compiling them.

Existing approaches to lifted inference have primarily focused on standard probabilistic inference algorithms such as variable elimination (e.g., [Poole, 2003; de Salvo Braz *et al.*, 2005; Milch *et al.*, 2008]) and belief propagation (e.g., [Singla and Domingos, 2008]). WFOMC differs from these types of methods in two significant ways. First, we take a model theoretic approach to inference. Second, our algorithm exploits *local structure* (e.g., context specific independence) within the the model. Considering local structure can substantially improve the efficiency of inference, and this type of structure is common in probabilistic logical models.

The approaches of Jha *et al.* [2010] and Gogate and Domingos [2010] are more similar in spirit to our approach. Jha *et al.* also approach the problem from the perspective

[1] http://dtai.cs.kuleuven.be/wfomc/

[2] http://people.csail.mit.edu/milch/blog/

[3] In theory, C-FOVE should be able to solve this theory by grounding it and performing VE.

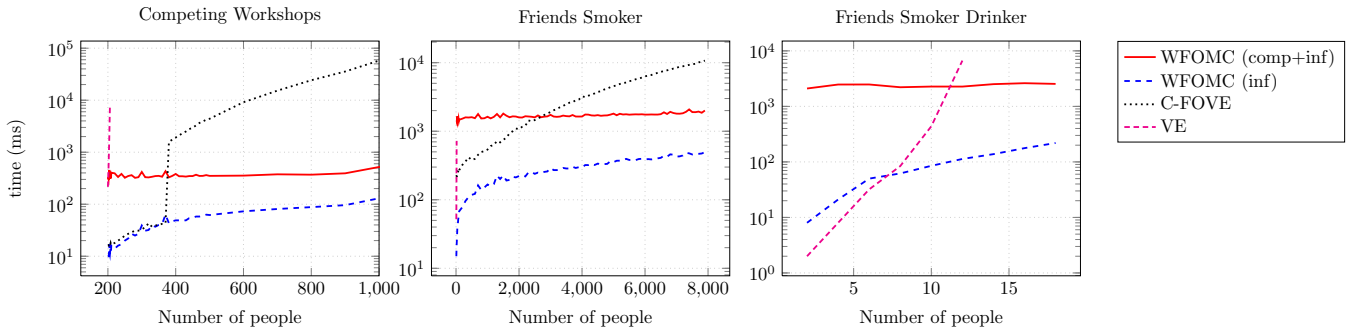


Figure 5: Inference times for the competing workshops, friends smokers, and friends smokers drinker tasks.

of logical inference and identify a limited class of models where tractable lifted inference is possible. For example, their approach excludes models that contain a clause where the same predicate appears more than once. Furthermore, it only works if the inference can be done fully on the lifted level. Gogate and Domingos [2010] propose a lifted version of AND/OR search. The d-DNNF structure is closely related to AND/OR structures. However, as noted by Dechter and Mateescu [2007], d-DNNF's are theoretically more expressive. Gogate and Domingos' approach differs in that they approach the problem from a different angle and use weighted CNF clauses instead of weighted model counting. Furthermore, they do not support caching and have no operator equivalent to the inclusion-exclusion operator.

6 Conclusions

We proposed the first model theoretic approach to lifted probabilistic inference. We introduced first-order d-DNNF circuits and presented an algorithm to compile a first-order probabilistic theory into one of these circuits. Our algorithm exploits well known concepts from logical inference including model counting, unit propagation, and Shannon decomposition. Furthermore, we drew connections between our model theoretic inference rules and those proposed for probabilistic inference, bringing us closer to understanding the connection between lifted inference in first-order logic and graphical models.

Acknowledgements The authors would like to thank Maurice Bruynooghe for valuable feedback. Guy Van den Broeck is supported by the Research Foundation-Flanders (FWO-Vlaanderen). This research is supported by GOA/08/008 'Probabilistic Logic Learning'.

References

[Boutilier *et al.*, 1996] Craig Boutilier, Nir Friedman, Moises Goldszmidt, and Daphne Koller. Context-specific independence in Bayesian networks. In *Proceedings of UAI*, pages 115–123, 1996.

[Chavira and Darwiche, 2008] Mark Chavira and Adnan Darwiche. On Probabilistic Inference by Weighted Model Counting. *Artificial Intelligence*, 172(6-7):772–799, April 2008.

[Chavira *et al.*, 2006] Mark Chavira, Adnan Darwiche, and Manfred Jaeger. Compiling Relational Bayesian Networks for Exact Inference. *International Journal of Approximate Reasoning*, 42(1-2):4–20, May 2006.

[Darwiche and Marquis, 2002] Adnan Darwiche and Pierre Marquis. A Knowledge Compilation Map. *Journal of Artificial Intelligence Research*, 17(1):229–264, 2002.

[De Raedt *et al.*, 2008] Luc De Raedt, Paolo Frasconi, Kristian Kersting, and Stephen Muggleton, editors. *Probabilistic inductive logic programming: theory and applications*. Springer-Verlag, Berlin, Heidelberg, 2008.

[de Salvo Braz *et al.*, 2005] Rodrigo de Salvo Braz, Eyal Amir, and Dan Roth. Lifted first-order probabilistic inference. In *Proceedings of IJCAI*, pages 1319–1325, 2005.

[Dechter and Mateescu, 2007] Rina Dechter and Robert Mateescu. AND/OR search spaces for graphical models. *Artificial Intelligence*, 171(2-3):73–106, February 2007.

[Getoor and Taskar, 2007] L. Getoor and B. Taskar, editors. *An Introduction to Statistical Relational Learning*. MIT Press, 2007.

[Gogate and Domingos, 2010] Vibhav Gogate and Pedro Domingos. Exploiting Logical Structure in Lifted Probabilistic Inference. In *Proceedings of StarAI*, 2010.

[Jha *et al.*, 2010] Abhay Jha, Vibhav Gogate, Alexandra Meliou, and Dan Suciu. Lifted Inference Seen from the Other Side: The Tractable Features. In *Proceedings of NIPS*, 2010.

[Kisyński and Poole, 2009] Jacek Kisyński and David Poole. Constraint processing in lifted probabilistic inference. In *Proceedings of UAI*, pages 293–302, 2009.

[Milch *et al.*, 2008] Brian Milch, Luke S. Zettlemoyer, Kristian Kersting, Michael Haimes, and Leslie Pack Kaelbling. Lifted Probabilistic Inference with Counting Formulas. In *Proceedings of AAAI*, pages 1062–1068, 2008.

[Poole, 2003] David Poole. First-order probabilistic inference. In *Proceedings of IJCAI*, pages 985–991, 2003.

[Singla and Domingos, 2008] P. Singla and P. Domingos. Lifted first-order belief propagation. In *Proceedings of AAAI*, pages 1094–1099, 2008.